Ontology and Database Systems: Knowledge Representation and Ontologies
Part 5: Reasoning in the DL-Lite Family

Diego Calvanese

Faculty of Computer Science
European Master in Computational Logic

A.Y. 2016/2017
Part 5

Reasoning in the $DL$-Lite family
Outline of Part 5

1. TBox reasoning
   - Preliminaries
   - Reducing to subsumption
   - Reducing to ontology unsatisfiability

2. TBox & ABox reasoning and query answering
   - TBox & ABox Reasoning services
   - Query answering
   - Query answering over satisfiable ontologies
   - Ontology satisfiability
   - Complexity of reasoning in DL-Lite

3. Beyond DL-Lite
   - Data complexity of query answering in DLs beyond DL-Lite
   - LOGSPACE-hard DLs
   - PTIME-hard DLs
   - coNP-hard DLs
   - Combining functionality and role inclusions
   - Unique name assumption
Outline of Part 5

1. TBox reasoning
   - Preliminaries
   - Reducing to subsumption
   - Reducing to ontology unsatisfiability

2. TBox & ABox reasoning and query answering

3. Beyond DL-Lite
Outline of Part 5

1. TBox reasoning
   - Preliminaries
     - Reducing to subsumption
     - Reducing to ontology unsatisfiability

2. TBox & ABox reasoning and query answering

3. Beyond DL-Lite
Remarks

In the following, we make some simplifying assumptions:

- We ignore the distinction between objects and values, since it is not relevant for reasoning. Hence we do not use value domains and attributes.
- We do not consider identification constraints.

Notation:

- When the distinction between $DL$-$Lite_R$, $DL$-$Lite_F$, or $DL$-$Lite_A$ is not important, we use just $DL$-$Lite$.
- $Q$ denotes a basic role, i.e., $Q \rightarrow P \mid P^\perp$.
- $R$ denotes a general role, i.e., $R \rightarrow Q \mid \neg Q$.
- $C$ denotes a general concept, i.e., $C \rightarrow A \mid \neg A \mid \exists Q \mid \neg \exists Q$, where $A$ is an atomic concept.
**TBox Reasoning services**

- **Concept Satisfiability:** A concept $C$ is satisfiable with respect to $\mathcal{T}$, if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^\mathcal{I}$ is not empty, i.e., $\mathcal{T} \not\models C \equiv \bot$.

- **Subsumption:** A concept $C_1$ is subsumed by a concept $C_2$ with respect to $\mathcal{T}$, if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^\mathcal{I} \subseteq C_2^\mathcal{I}$, i.e., $\mathcal{T} \models C_1 \sqsubseteq C_2$.

- **Equivalence:** Concepts $C_1$ and $C_2$ are equivalent with respect to $\mathcal{T}$, if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^\mathcal{I} = C_2^\mathcal{I}$, i.e., $\mathcal{T} \models C_1 \equiv C_2$.

- **Disjointness:** Concepts $C_1$ and $C_2$ are disjoint with respect to $\mathcal{T}$, if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^\mathcal{I} \cap C_2^\mathcal{I} = \emptyset$, i.e., $\mathcal{T} \models C_1 \sqcap C_2 \equiv \bot$.

- **Functionality implication:** A functionality assertion $(\text{funct } Q)$ is logically implied by $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$, we have that $(o, o_1) \in Q^\mathcal{I}$ and $(o, o_2) \in Q^\mathcal{I}$ implies $o_1 = o_2$, i.e., $\mathcal{T} \models (\text{funct } Q)$.

Analogous definitions hold for role satisfiability, subsumption, equivalence, and disjointness.
Basic reasoning service:

- **Ontology satisfiability**: Verify whether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.

In the following, we show how to reduce TBox reasoning to ontology unsatisfiability:

1. We show how to reduce TBox reasoning services to concept/role subsumption.
2. We provide reductions from concept/role subsumption to ontology unsatisfiability.
Outline of Part 5

1. **TBox reasoning**
   - Preliminaries
   - Reducing to subsumption
   - Reducing to ontology unsatisfiability

2. **TBox & ABox reasoning and query answering**

3. **Beyond DL-Lite**
Concept/role satisfiability, equivalence, and disjointness

**Theorem**

1. $C$ is unsatisfiable wrt $\mathcal{T}$ iff $\mathcal{T} \models C \sqsubseteq \neg C$.
2. $\mathcal{T} \models C_1 \equiv C_2$ iff $\mathcal{T} \models C_1 \sqsubseteq C_2$ and $\mathcal{T} \models C_2 \sqsubseteq C_1$.
3. $C_1$ and $C_2$ are disjoint iff $\mathcal{T} \models C_1 \sqsubseteq \neg C_2$.

**Proof (sketch).**

1. “$\Leftarrow$” if $\mathcal{T} \models C \sqsubseteq \neg C$, then $C^\mathcal{I} \subseteq \Delta^\mathcal{I} \setminus C^\mathcal{I}$, for every model $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$ of $\mathcal{T}$; but this holds iff $C^\mathcal{I} = \emptyset$.
   “$\Rightarrow$” if $C$ is unsatisfiable, then $C^\mathcal{I} = \emptyset$, for every model $\mathcal{I}$ of $\mathcal{T}$. Therefore $C^\mathcal{I} \subseteq (\neg C')^\mathcal{I}$.

2. Trivial.

3. Trivial.

Analogous reductions for role satisfiability, equivalence and disjointness.
From implication of functionalities to subsumption

Theorem

\[ \mathcal{T} \models (\text{funct } Q) \text{ iff either} \]
- \( (\text{funct } Q) \in \mathcal{T} \) (only for DL-Lite\(_F\) or DL-Lite\(_A\)), or
- \( \mathcal{T} \models Q \subseteq \neg Q \).

Proof (sketch).

“\( \Leftarrow \)” The case in which \((\text{funct } Q) \in \mathcal{T}\) is trivial. Instead, if \( \mathcal{T} \models Q \subseteq \neg Q \), then \( Q^\mathcal{I} = \emptyset \) and hence \( \mathcal{I} \models (\text{funct } Q) \), for every model \( \mathcal{I} \) of \( \mathcal{T} \).

“\( \Rightarrow \)” When neither \((\text{funct } Q) \in \mathcal{T}\) nor \( \mathcal{T} \models Q \subseteq \neg Q \), we can construct a model of \( \mathcal{T} \) that is not a model of \((\text{funct } Q)\).

The interesting part of this result is the “only-if” direction, telling us that in DL-Lite functionality is implied only in trivial ways.
Outline of Part 5

1. **TBox reasoning**
   - Preliminaries
   - Reducing to subsumption
   - Reducing to ontology unsatisfiability

2. **TBox & ABox reasoning and query answering**

3. **Beyond DL-Lite**
From concept subsumption to ontology unsatisfiability

Theorem

\[ T \models C_1 \sqsubseteq C_2 \text{ iff the ontology } O_{C_1 \sqsubseteq C_2} = \langle T \cup \{ \hat{A} \sqsubseteq C_1, \hat{A} \sqsubseteq \neg C_2 \}, \{ \hat{A}(c) \} \rangle \text{ is unsatisfiable}, \]

where \( \hat{A} \) is an atomic concept not in \( T \), and \( c \) is a constant.

Intuitively, \( C_1 \) is subsumed by \( C_2 \) iff the smallest ontology containing \( T \) and implying both \( C_1(c) \) and \( \neg C_2(c) \) is unsatisfiable.

Proof (sketch).

“\( \Leftarrow \)” Let \( O_{C_1 \sqsubseteq C_2} \) be unsatisfiable, and suppose that \( T \not\models C_1 \sqsubseteq C_2 \). Then there exists a model \( I \) of \( T \) such that \( C_1^T \not\subseteq C_2^T \). Hence \( C_1^T \setminus C_2^T \neq \emptyset \). We can extend \( I \) to a model of \( O_{C_1 \sqsubseteq C_2} \) by taking \( c^I = o \), for some \( o \in C_1^T \setminus C_2^T \), and \( \hat{A}^I = \{ c^I \} \). This contradicts \( O_{C_1 \sqsubseteq C_2} \) being unsatisfiable.

“\( \Rightarrow \)” Let \( T \models C_1 \sqsubseteq C_2 \), and suppose that \( O_{C_1 \sqsubseteq C_2} \) is satisfiable. Then there exists a model \( I \) be of \( O_{C_1 \sqsubseteq C_2} \). Then \( I \models T \), and \( I \models C_1(c) \) and \( I \models \neg C_2(c) \), i.e., \( I \not\models C_1 \sqsubseteq C_2 \). This contradicts \( T \models C_1 \sqsubseteq C_2 \).
From role subsumption to ont. unsatisfiability for \( DL-Lite_\mathcal{R} \)

**Theorem**

Let \( \mathcal{T} \) be a \( DL-Lite_\mathcal{R} \) or \( DL-Lite_\mathcal{A} \) TBox and \( R_1, R_2 \) two general roles. Then \( \mathcal{T} \models R_1 \subseteq R_2 \) iff the ontology \( \mathcal{O}_{R_1 \subseteq R_2} = \langle \mathcal{T} \cup \{\hat{P} \subseteq R_1, \hat{P} \subseteq \neg R_2\}, \{\hat{P}(c_1, c_2)\} \rangle \) is unsatisfiable, where \( \hat{P} \) is an atomic role not in \( \mathcal{T} \), and \( c_1, c_2 \) are two constants.

Intuitively, \( R_1 \) is subsumed by \( R_2 \) iff the smallest ontology containing \( \mathcal{T} \) and implying both \( R_1(c_1, c_2) \) and \( \neg R_2(c_1, c_2) \) is unsatisfiable.

**Proof (sketch).**

Analogous to the one for concept subsumption.

Notice that \( \mathcal{O}_{R_1 \subseteq R_2} \) is inherently a \( DL-Lite_\mathcal{R} \) (or \( DL-Lite_\mathcal{A} \)) ontology.
From role subsumption to ont. unsatisfiability for $DL\text{-Lite}_F$

**Theorem**

Let $\mathcal{T}$ be a $DL\text{-Lite}_F$ TBox, and $Q_1$, $Q_2$ two basic roles such that $Q_1 \neq Q_2$. Then,

1. $\mathcal{T} \models Q_1 \sqsubseteq Q_2$ iff $Q_1$ is unsatisfiable iff
   either $\exists Q_1$ or $\exists Q_1^-$ is unsatisfiable wrt $\mathcal{T}$,
   which can again be reduced to ontology unsatisfiability.

2. $\mathcal{T} \models \neg Q_1 \sqsubseteq Q_2$ iff $\mathcal{T}$ is unsatisfiable (which is never the case for $DL\text{-Lite}$).

3. $\mathcal{T} \models Q_1 \sqsubseteq \neg Q_2$ iff
   either $\exists Q_1$ and $\exists Q_2$ are disjoint, or $\exists Q_1^-$ and $\exists Q_2^-$ are disjoint, iff
   either $\mathcal{T} \models \exists Q_1 \sqsubseteq \neg \exists Q_2$, or $\mathcal{T} \models \exists Q_1^- \sqsubseteq \neg \exists Q_2^-$,
   which can again be reduced to ontology unsatisfiability.

Notice that an inclusion of the form $\neg Q_1 \sqsubseteq \neg Q_2$ is equivalent to $Q_2 \sqsubseteq Q_1$, and therefore is considered in the first item.
The results above tell us that we can support TBox reasoning services by relying on the ontology (un)satisfiability service.

Ontology satisfiability is a form of reasoning over both the TBox and the ABox of the ontology.

In the following, we first consider other TBox & ABox reasoning services, in particular query answering, and then turn back to ontology satisfiability.
Outline of Part 5

1. TBox reasoning

2. TBox & ABox reasoning and query answering
   - TBox & ABox Reasoning services
   - Query answering
   - Query answering over satisfiable ontologies
   - Ontology satisfiability
   - Complexity of reasoning in DL-Lite

3. Beyond DL-Lite
Outline of Part 5

1. TBox reasoning

2. TBox & ABox reasoning and query answering
   - TBox & ABox Reasoning services
     - Query answering
     - Query answering over satisfiable ontologies
     - Ontology satisfiability
     - Complexity of reasoning in DL-Lite

3. Beyond DL-Lite
TBox and ABox reasoning services

- **Ontology Satisfiability**: Verify whether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.

- **Concept Instance Checking**: Verify whether an individual $c$ is an instance of a concept $C$ in an ontology $\mathcal{O}$, i.e., whether $\mathcal{O} \models C(c)$.

- **Role Instance Checking**: Verify whether a pair $(c_1, c_2)$ of individuals is an instance of a role $R$ in an ontology $\mathcal{O}$, i.e., whether $\mathcal{O} \models R(c_1, c_2)$.

- **Query Answering**: Given a query $q$ over an ontology $\mathcal{O}$, find all tuples $\overrightarrow{c}$ of constants such that $\mathcal{O} \models q(\overrightarrow{c})$. 
Query answering and instance checking

For atomic concepts and roles, **instance checking is a special case of query answering**, in which the query is **boolean** and constituted by a single atom in the body.

- $\mathcal{O} \models A(c)$ iff $q() \leftarrow A(c)$ evaluated over $\mathcal{O}$ is true.
- $\mathcal{O} \models P(c_1, c_2)$ iff $q() \leftarrow P(c_1, c_2)$ evaluated over $\mathcal{O}$ is true.
From instance checking to ontology unsatisfiability

Theorem

Let $O = \langle T, A \rangle$ be a $DL$-Lite ontology, $C$ a $DL$-Lite concept, and $P$ an atomic role. Then:

- $O \models C(c) \iff O_{C(c)} = \langle T \cup \{ \hat{A} \sqsubseteq \neg C \}, A \cup \{ \hat{A}(c) \} \rangle$ is unsatisfiable, where $\hat{A}$ is an atomic concept not in $O$.
- $O \models \neg P(c_1, c_2) \iff O_{\neg P(c_1, c_2)} = \langle T, A \cup \{ P(c_1, c_2) \} \rangle$ is unsatisfiable.

Theorem

Let $O = \langle T, A \rangle$ be a $DL$-Lite$_F$ ontology and $P$ an atomic role. Then $O \models P(c_1, c_2)$ iff $O$ is unsatisfiable or $P(c_1, c_2) \in A$.

Theorem

Let $O = \langle T, A \rangle$ be a $DL$-Lite$_R$ or $DL$-Lite$_A$ ontology and $P$ an atomic role. Then $O \models P(c_1, c_2)$ iff $O_{P(c_1, c_2)} = \langle T \cup \{ \hat{P} \sqsubseteq \neg P \}, A \cup \{ \hat{P}(c_1, c_2) \} \rangle$ is unsatisfiable, where $\hat{P}$ is an atomic role not in $O$. 
Outline of Part 5

1. TBox reasoning
2. TBox & ABox reasoning and query answering
   - TBox & ABox Reasoning services
   - Query answering
     - Query answering over satisfiable ontologies
     - Ontology satisfiability
     - Complexity of reasoning in DL-Lite
3. Beyond DL-Lite
Certain answers

We recall that

Query answering over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a form of **logical implication**:

$$\text{find all tuples } \vec{c} \text{ of constants of } \mathcal{A} \text{ s.t. } \mathcal{O} \models q(\vec{c})$$

A.k.a. **certain answers** in databases, i.e., the tuples that are answers to $q$ in **all** models of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$:

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

**Note:** We have assumed that the answer $q^\mathcal{I}$ to a query $q$ over an interpretation $\mathcal{I}$ is constituted by a set of tuples of **constants** of $\mathcal{A}$, rather than objects in $\Delta^\mathcal{I}$.
Q-rewritability for DL-Lite

- We now study rewritability of query answering over DL-Lite ontologies.

- In particular we will show that DL-Lite\(_A\) (and hence DL-Lite\(_F\) and DL-Lite\(_R\)) enjoy FOL-rewritability of answering union of conjunctive queries.
Query answering vs. ontology satisfiability

- In the case in which an ontology is unsatisfiable, according to the “ex falso quod libet” principle, reasoning is trivialized.
- In particular, query answering is meaningless, since every tuple is in the answer to every query.
- We are not interested in encoding meaningless query answering into the perfect reformulation of the input query. Therefore, before query answering, we will always check ontology satisfiability to single out meaningful cases.

Thus, we proceed as follows:
1. We show how to do query answering over satisfiable ontologies.
2. We show how we can exploit the query answering algorithm also to check ontology satisfiability.
Positive vs. negative inclusions

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

\[
\begin{align*}
& A_1 \sqsubseteq A_2 \\
& A_1 \sqsubseteq \exists Q_2 \\
& \exists Q_1 \sqsubseteq A_2 \\
& \exists Q_1 \sqsubseteq \exists Q_2
\end{align*}
\]

We call **negative inclusions (NIs)** assertions of the form

\[
\begin{align*}
& A_1 \sqsubseteq \neg A_2 \\
& A_1 \sqsubseteq \neg \exists Q_2 \\
& \exists Q_1 \sqsubseteq \neg A_2 \\
& \exists Q_1 \sqsubseteq \neg \exists Q_2
\end{align*}
\]
Outline of Part 5

1. TBox reasoning

2. TBox & ABox reasoning and query answering
   - TBox & ABox Reasoning services
   - Query answering
   - Query answering over satisfiable ontologies
     - Ontology satisfiability
     - Complexity of reasoning in DL-Lite

3. Beyond DL-Lite
Query answering over satisfiable ontologies

Given a CQ $q$ and a satisfiable ontology $O = \langle T, A \rangle$, we compute $cert(q, O)$ as follows:

1. **Using $T$, rewrite $q$ into a UCQ $r_{q,T}$** (the perfect rewriting of $q$ w.r.t. $T$).
2. **Evaluate $r_{q,T}$ over $A$** (simply viewed as data), to return $cert(q, O)$.

Correctness of this procedure shows FOL-rewritability of query answering in $DL-Lite$. 
Query rewriting step: Basic idea

Intuition: a positive inclusion corresponds to a logic programming rule.

Basic rewriting step:

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

We say that the positive inclusion applies to the atom.

Example

The positive inclusion corresponds to the logic programming rule: \( \text{AssistantProf} \sqsubseteq \text{Professor} \)

Consider the query: \( q(x) \leftarrow \text{Professor}(x) \).

By applying the positive inclusion to the atom \( \text{Professor}(x) \), we generate:

\( q(x) \leftarrow \text{AssistantProf}(x) \).

This query is added to the input query, and contributes to the perfect rewriting.
Query rewriting (cont’d)

Example

Consider the query

\[ q(x) \leftarrow teaches(x, y), Course(y) \]

and the PI

\[ \exists teaches \sqsubseteq Course \]

as a logic programming rule:

\[ Course(z_2) \leftarrow teaches(z_1, z_2). \]

The PI applies to the atom \( Course(y) \), and we add to the perfect rewriting the query

\[ q(x) \leftarrow teaches(x, y), teaches(z_1, y). \]

Example

Consider now the query

\[ q(x) \leftarrow teaches(x, y) \]

and the PI

\[ Professor \sqsubseteq \exists teaches \]

as a logic programming rule:

\[ teaches(z, f(z)) \leftarrow Professor(z). \]

The PI applies to the atom \( teaches(x, y) \), and we add to the perfect rewriting the query

\[ q(x) \leftarrow Professor(x). \]
Query rewriting – Constants

Example

Conversely, for the query

\[ q(x) \leftarrow teaches(x, kr) \]

and the same PI as before

\[ Professor \sqsubseteq \exists teaches \]

as a logic programming rule:

\[ teaches(z, f(z)) \leftarrow Professor(z) \]

\( teaches(x, kr) \) does not unify with \( teaches(z, f(z)) \), since the skolem term \( f(z) \) in the head of the rule does not unify with the constant \( kr \).

Remember: We adopt the unique name assumption.

In this case, we say that the PI does not apply to the atom \( teaches(x, kr) \).

Example

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

\[ q(x, y) \leftarrow teaches(x, y). \]
Query rewriting – Join variables

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

**Example**

Consider the query

\[ q(x) \leftarrow teaches(x, y), \text{Course}(y) \]

and the PI

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

as a logic programming rule:

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

The PI above does **not** apply to the atom \( teaches(x, y) \).
Query rewriting – Reduce step

Example

Consider now the query \( q(x) \leftarrow teaches(x, y), teaches(z, y) \)

and the PI \( \text{Professor} \sqsubseteq \exists \text{teaches} \)
as a logic rule: \( teaches(z, f(z)) \leftarrow \text{Professor}(z) \).

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

Example

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

\[
q(x) \leftarrow teaches(x, y).
\]

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

\[
q(x) \leftarrow \text{Professor}(x).
\]
Query rewriting – Summary

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

1. **Apply** to some atom of $q'$ a PI in $\mathcal{T}$ as follows:

   $A_1 \sqsubseteq A_2 \quad \ldots, A_2(x), \ldots \rightarrow \ldots, A_1(x), \ldots$

   $\exists P \sqsubseteq A \quad \ldots, A(x), \ldots \rightarrow \ldots, P(x, -), \ldots$

   $\exists P^- \sqsubseteq A \quad \ldots, A(x), \ldots \rightarrow \ldots, P(-, x), \ldots$

   $A \sqsubseteq \exists P \quad \ldots, P(x, -), \ldots \rightarrow \ldots, A(x), \ldots$

   $A \sqsubseteq \exists P^- \quad \ldots, P(-, x), \ldots \rightarrow \ldots, A(x), \ldots$

   $\exists P_1 \sqsubseteq \exists P_2 \quad \ldots, P_2(x, -), \ldots \rightarrow \ldots, P_1(x, -), \ldots$

   $P_1 \sqsubseteq P_2 \quad \ldots, P_2(x, y), \ldots \rightarrow \ldots, P_1(x, y), \ldots$

   $P_1 \sqsubseteq P^- \quad \ldots, P_2(x, y), \ldots \rightarrow \ldots, P_1(y, x), \ldots$

   $\ldots$

   (‘-’ denotes an **unbound** variable, i.e., a variable that appears only once)

2. Choose two atoms of $q'$ that unify, and **apply the unifier** to $q'$.

Each time, the result of the above step is added to the queries to be processed.

**Note:** Unifying atoms can make rules applicable that were not so before, and is required for completeness of the method.

The UCQ resulting from this process is the **perfect rewriting** $r_{q, \mathcal{T}}$. 
Query rewriting algorithm

**Algorithm** PerfectRef\((Q, T_P)\)

**Input:** union of conjunctive queries \(Q\), set of DL-Lite\(A\) PIs \(T_P\)

**Output:** union of conjunctive queries \(PR\)

\[ PR := Q; \]

repeat

\[ PR' := PR; \]

for each \(q \in PR'\) do

for each \(g\) in \(q\) do

for each PI \(I\) in \(T_P\) do

if \(I\) is applicable to \(g\) then \(PR := PR \cup \{ ApplyPI(q, g, I) \}\);

for each \(g_1, g_2\) in \(q\) do

if \(g_1\) and \(g_2\) unify then \(PR := PR \cup \{ \tau(Reduce(q, g_1, g_2)) \}\);

until \(PR' = PR\);

return \(PR\)

**Observations:**

- Termination follows from having only finitely many different rewritings.
- NIs or functionalities do not play any role in the rewriting of the query.
Query answering in *DL-Lite* – Example

**TBox:**

\[
\begin{align*}
\text{AssistantProf} & \sqsubseteq \text{Professor} \\
\text{Professor} & \sqsubseteq \exists \text{teaches} \\
\exists \text{teaches}^- & \sqsubseteq \text{Course}
\end{align*}
\]

**Corresponding rules:**

\[
\begin{align*}
\text{AssistantProf}(x) & \rightarrow \text{Professor}(x) \\
\text{Professor}(x) & \rightarrow \exists y(\text{teaches}(x, y)) \\
\text{teaches}(y, x) & \rightarrow \text{Course}(x)
\end{align*}
\]

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

**Perfect rewriting:**

\[
\begin{align*}
q(x) & \leftarrow \text{teaches}(x, y), \text{Course}(y) \\
q(x) & \leftarrow \text{teaches}(x, y), \text{teaches}(\_ , y) \\
q(x) & \leftarrow \text{teaches}(x, \_) \\
q(x) & \leftarrow \text{Professor}(x) \\
q(x) & \leftarrow \text{AssistantProf}(x)
\end{align*}
\]

**ABox:**

\[
\begin{align*}
\text{teaches}(\text{john}, \text{kr}) & \quad \text{AssistantProf}(\text{john}) \\
\text{teaches}(\text{tim}, \text{db}) & \quad \text{AssistantProf}(\text{mary})
\end{align*}
\]

Evaluating the perfect rewriting over the ABox (seen as a DB) produces as answer \{\text{john}, \text{tim}, \text{mary}\}.
Query answering in **DL-Lite** – An interesting example

**TBox:**  
\[ \text{Person} \sqsubseteq \exists \text{hasFather} \]  
\[ \exists \text{hasFather}^- \sqsubseteq \text{Person} \]

**ABox:**  
\[ \text{Person}(\text{mary}) \]

**Query:**  
\[ q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, y_3) \]

\[ q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, \_)) \]
\[ \text{Apply } \text{Person} \sqsubseteq \exists \text{hasFather} \text{ to the atom } \text{hasFather}(y_2, \_) \]
\[ q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{Person}(y_2) \]
\[ \text{Apply } \exists \text{hasFather}^- \sqsubseteq \text{Person} \text{ to the atom } \text{Person}(y_2) \]
\[ q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(\_, y_2) \]
\[ \text{Unify atoms } \text{hasFather}(y_1, y_2) \text{ and } \text{hasFather}(\_, y_2) \]
\[ q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2) \]
\[ \vdots \]
\[ q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, \_) \]
\[ \text{Apply } \text{Person} \sqsubseteq \exists \text{hasFather} \text{ to the atom } \text{hasFather}(x, \_) \]
\[ q(x) \leftarrow \text{Person}(x) \]
Query answering over satisfiable **DL-Lite** ontologies

For an ABox $\mathcal{A}$ and a query $q$ over $\mathcal{A}$, let $\text{Eval}_{\text{CWA}}(q, \mathcal{A})$ denote the evaluation of $q$ over $\mathcal{A}$ considered as a database (i.e., considered under the CWA).

**Theorem**

Let $\mathcal{T}$ be a **DL-Lite** TBox, $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$, and $q$ a CQ over $\mathcal{T}$. Then, for each ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, we have that

$$\text{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \text{Eval}_{\text{CWA}}(\text{PerfectRef}(q, \mathcal{T}_P), \mathcal{A}).$$

As a consequence, query answering over a satisfiable **DL-Lite** ontology is FOL-rewritable.

Notice that we did not use NIs or functionality assertions of $\mathcal{T}$ in computing $\text{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle)$. Indeed, when the ontology is satisfiable, we can ignore NIs and functionality assertions for query answering.
Canonical model of a *DL-Lite* ontology

The proof of the previous result exploits a fundamental property of *DL-Lite*, that relies on the following notion.

**Def.: Canonical model**

Let $O = \langle T, A \rangle$ be a *DL-Lite* ontology. A model $I_O$ of $O$ is called **canonical** if for every model $I$ of $O$ there is a homomorphism from $I_O$ to $I$.

**Theorem**

Every satisfiable *DL-Lite* ontology has a **canonical model**.

Properties of the canonical models of a *DL-Lite* ontology:

- A canonical model is in general infinite.
- All canonical models are homomorphically equivalent, hence we can do as if there was a single canonical model.
Query answering in *DL-Lite* – Canonical model

From the definition of canonical model, and since homomorphisms are closed under composition, we get that:

To compute the certain answer to a query $q$ over an ontology $\mathcal{O}$, one could in principle evaluate $q$ over a canonical model $\mathcal{I}_\mathcal{O}$ of $\mathcal{O}$.

- This does not give us directly an algorithm for query answering over an ontology $\mathcal{O} = \langle T, A \rangle$, since $\mathcal{I}_\mathcal{O}$ may be infinite.
- However, one can show that evaluating $q$ over $\mathcal{I}_\mathcal{O}$ amounts to evaluating the perfect rewriting $r_{q,T}$ over $A$. 
Using RDBMS technology for query answering

The **ABox** $\mathcal{A}$ can be stored as a **relational database** in a standard RDBMS:

- For each atomic concept $A$ of the ontology:
  - define a **unary relational table** $\text{tab}_A$,
  - populate $\text{tab}_A$ with each $\langle c \rangle$ such that $A(c) \in \mathcal{A}$.
- For each atomic role $P$ of the ontology,
  - define a **binary relational table** $\text{tab}_P$,
  - populate $\text{tab}_P$ with each $\langle c_1, c_2 \rangle$ such that $P(c_1, c_2) \in \mathcal{A}$.

We have that query answering over satisfiable **DL-Lite** ontologies can be done effectively using RDBMS technology:

$$\text{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \text{Eval}(\text{SQL}(\text{PerfectRef}(q, \mathcal{T}_P)), \text{DB}(\mathcal{A}))$$

Where:
- $\text{Eval}(q_s, DB)$ denotes the evaluation of an SQL query $q_s$ over a database $DB$.
- $\text{SQL}(q)$ denotes the SQL encoding of a UCQ $q$.
- $\text{DB}(\mathcal{A})$ denotes the database obtained as above.
Outline of Part 5

1. TBox reasoning

2. TBox & ABox reasoning and query answering
   - TBox & ABox Reasoning services
   - Query answering
   - Query answering over satisfiable ontologies
   - Ontology satisfiability
   - Complexity of reasoning in DL-Lite

3. Beyond DL-Lite
Satisfiability of ontologies with only PIs

Let us now consider the problem of establishing whether an ontology is satisfiable.

A first notable result tells us that inclusion assertions alone cannot give rise to unsatisfiable ontologies.

**Theorem**

Let $\mathcal{O} = \langle T, A \rangle$ be a $DL$- Literal ontology where $T$ contains only PIs. Then, $\mathcal{O}$ is satisfiable.
Satisfiability of $DL$-$Lite_A$ ontologies

Unsatisfiability in $DL$-$Lite_A$ ontologies can be caused by NIs or by functionality assertions.

Example

$TBox \ T$: $Professor \sqsubseteq \neg \text{Student}$

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

($\text{funct} \ \text{teaches}^-$)

$ABox \ A$: $\text{Student}(john)$

$\text{teaches}(john, kr)$

$\text{teaches}(michael, kr)$
Checking satisfiability of $DL\text{-}Lite_{\mathcal{A}}$ ontologies

Satisfiability of a $DL\text{-}Lite_{\mathcal{A}}$ ontology $O = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluating over $DB(\mathcal{A})$ a UCQ that asks for the existence of objects violating the NI or functionality assertions.

Let $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$.
We deal with NIs and functionality assertions differently.

For each NI $N \in \mathcal{T}$:
1. we construct a boolean CQ $q_N()$ such that
   $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$ iff $\langle \mathcal{T}_P \cup \{N\}, \mathcal{A} \rangle$ is unsatisfiable
2. We check whether $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$ using $PerfectRef$, i.e., we compute $PerfectRef(q_N, \mathcal{T}_P)$, and evaluate it over $DB(\mathcal{A})$.

For each functionality assertion $F \in \mathcal{T}$:
1. we construct a boolean CQ $q_F()$ such that
   $\mathcal{A} \models q_F()$ iff $\langle \{F\}, \mathcal{A} \rangle$ is unsatisfiable.
2. We check whether $\mathcal{A} \models q_F()$, by simply evaluating $q_F$ over $DB(\mathcal{A})$. 
Checking violations of negative inclusions

For each **NI** $N$ in $\mathcal{T}$ we compute a boolean CQ $q_N()$ according to the following rules:

\[
\begin{align*}
A_1 \sqsubseteq \neg A_2 & \quad \Rightarrow \quad q_N() \leftarrow A_1(x), A_2(x) \\
\exists P \sqsubseteq \neg A \text{ or } A \sqsubseteq \neg \exists P & \quad \Rightarrow \quad q_N() \leftarrow P(x, y), A(x) \\
\exists P^- \sqsubseteq \neg A \text{ or } A \sqsubseteq \neg \exists P^- & \quad \Rightarrow \quad q_N() \leftarrow P(y, x), A(x) \\
\exists P_1 \sqsubseteq \neg \exists P_2 & \quad \Rightarrow \quad q_N() \leftarrow P_1(x, y), P_2(x, z) \\
\exists P_1 \sqsubseteq \neg \exists P_2^- & \quad \Rightarrow \quad q_N() \leftarrow P_1(x, y), P_2(z, x) \\
\exists P_1^- \sqsubseteq \neg \exists P_2 & \quad \Rightarrow \quad q_N() \leftarrow P_1(x, y), P_2(y, z) \\
\exists P_1^- \sqsubseteq \neg \exists P_2^- & \quad \Rightarrow \quad q_N() \leftarrow P_1(x, y), P_2(z, y) \\
P_1 \sqsubseteq \neg P_2 \text{ or } P_1^- \sqsubseteq \neg P_2^- & \quad \Rightarrow \quad q_N() \leftarrow P_1(x, y), P_2(x, y) \\
P_1^- \sqsubseteq \neg P_2 \text{ or } P_1 \sqsubseteq \neg P_2^- & \quad \Rightarrow \quad q_N() \leftarrow P_1(x, y), P_2(y, x)
\end{align*}
\]
Checking violations of negative inclusions – Example

PIs $\mathcal{T}_P$:

$\exists$teaches $\sqsubseteq$ Professor

NIs $\mathcal{N}$:

Professor $\sqsubseteq \neg$Student

Query $q_N$:

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

Perfect Rewriting:

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

$q_N() \leftarrow \text{Student}(x), \text{teaches}(x, \_)$

ABox $\mathcal{A}$:

$\text{teaches}(\text{john}, \text{kr})$

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

It is easy to see that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$, and that the ontology $\langle \mathcal{T}_P \cup \{\text{Professor} \sqsubseteq \neg\text{Student}\}, \mathcal{A} \rangle$ is unsatisfiable.
Boolean queries vs. non-boolean queries for NIs

To ensure correctness of the method, the queries that check for the violation of a NI need to be **boolean**.

Example

TBox $\mathcal{T}$: 

- $A_1 \sqsubseteq \neg A_0$
- $A_1 \sqsubseteq A_0$
- $\exists P \subseteq A_1$
- $A_2 \subseteq \exists P$

ABox $\mathcal{A}$: $A_2(c)$

Since $A_1$, $P$, and $A_2$ are unsatisfiable, also $\langle \mathcal{T}, \mathcal{A} \rangle$ is **unsatisfiable**.

Consider the query corresponding to the NI $A_1 \sqsubseteq \neg A_0$.

$q_N() \leftarrow A_1(x), A_0(x)$

Then $\text{PerfectRef}(q_N, \mathcal{T}_P)$ is:

$q_N() \leftarrow A_1(x), A_0(x)$
$q_N() \leftarrow A_1(x)$
$q_N() \leftarrow P(x, \_)$
$q_N() \leftarrow A_2(\_)$

We have that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$.

$q'_N(x) \leftarrow A_1(x), A_0(x)$

Then $\text{PerfectRef}(q'_N, \mathcal{T}_P)$ is:

$q'_N(x) \leftarrow A_1(x), A_0(x)$
$q'_N(x) \leftarrow A_1(x)$
$q'_N(x) \leftarrow P(x, \_)$

$\text{cert}(q'_N, \langle \mathcal{T}_P, \mathcal{A} \rangle) = \emptyset$, hence $q'_N(x)$ does not detect unsatisfiability.
Checking violations of functionality assertions

For each functionality assertion $F$ in $\mathcal{T}$ we compute a boolean FOL query $q_F()$ according to the following rules:

\[
\begin{align*}
\text{(funct } P) & \leadsto q_F() \leftarrow P(x, y), P(x, z), y \neq z \\
\text{(funct } P^{-}) & \leadsto q_F() \leftarrow P(x, y), P(z, y), x \neq z
\end{align*}
\]

Example

Functionality $F$: \(\text{(funct teaches}^{-})\)

Query $q_F$: \(q_F() \leftarrow \text{teaches}(x, y), \text{teaches}(z, y), x \neq z\)

ABox $\mathcal{A}$: \(\text{teaches(john, kr)}\) \n\(\text{teaches(michael, kr)}\)

It is easy to see that $\mathcal{A} \models q_F()$, and that \(\langle \{\text{(funct teaches}^{-})\}, \mathcal{A} \rangle\), is unsatisfiable.
From satisfiability to query answering in $DL$-$Lite_A$

**Lemma (Separation for $DL$-$Lite_A$)**

Let $O = \langle T, A \rangle$ be a $DL$-$Lite_A$ ontology, and $T_P$ the set of PIs in $T$. Then, $O$ is unsatisfiable if and only if one of the following conditions holds:

(a) There exists a NI $N \in T$ such that $\langle T_P, A \rangle \models q_N()$.

(b) There exists a functionality assertion $F \in T$ such that $A \models q_F()$.

(a) relies on the properties that **NIs do not interact with each other**, and that **interaction between NIs and PIs** is captured through **PerfectRef**.

(b) exploits the property that **NIs and PIs do not interact with functionalities**: indeed, no functionality assertion is contradicted in a $DL$-$Lite_A$ ontology $O$, beyond those explicitly contradicted by the ABox.

Notably, to check ontology satisfiability, each NI and each functionality assertion can be processed individually.
FOL-rewritability of satisfiability in $DL\text{-}Lite_\mathcal{A}$

From the previous lemma and the theorem on query answering for satisfiable $DL\text{-}Lite_\mathcal{A}$ ontologies, we get the following result.

**Theorem**

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a $DL\text{-}Lite_\mathcal{A}$ ontology, and $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$. Then, $\mathcal{O}$ is unsatisfiable iff one of the following condition holds:

(a) There exists a NI $N \in \mathcal{T}$ s.t. $Eval_{CWA}(\text{PerfectRef}(q_N, \mathcal{T}_P), \mathcal{A})$ returns $true$.

(b) There exists a func. assertion $F \in \mathcal{T}$ s.t. $Eval_{CWA}(q_F, \mathcal{A})$ returns $true$.

**Note:** All the queries $q_N()$ and $q_F()$ can be combined into a single UCQ. Hence, satisfiability of a $DL\text{-}Lite_\mathcal{A}$ ontology is reduced to evaluating a FOL-query over an ontology whose TBox consists of positive inclusions only (and hence is satisfiable).
Outline of Part 5

1. TBox reasoning

2. TBox & ABox reasoning and query answering
   - TBox & ABox Reasoning services
   - Query answering
   - Query answering over satisfiable ontologies
   - Ontology satisfiability
   - Complexity of reasoning in DL-Lite

3. Beyond DL-Lite
Complexity of query answering over satisfiable ontologies

Theorem

Query answering over $DL-Lite_A$ ontologies is

1. **NP-complete** in the size of query and ontology (combined complexity).
2. **PTIME** in the size of the ontology (schema+data complexity).
3. **AC$^0$** in the size of the ABox (data complexity).

Proof (sketch).

1. **Guess** together the derivation of one of the CQs of the perfect rewriting, and an assignment to its existential variables. Checking the derivation and evaluating the guessed CQ over the ABox is then polynomial in combined complexity. NP-hardness follows from combined complexity of evaluating CQs over a database.
2. The number of CQs in the perfect rewriting is polynomial in the size of the TBox, and we can compute them in **PTIME**.
3. **AC$^0$** is the data complexity of evaluating FOL queries over a DB.

Diego Calvanese (unibz)
**Complexity of ontology satisfiability**

**Theorem**

Checking satisfiability of $DL$-$Lite_A$ ontologies is

1. $\mathbf{PTIME}$ in the size of the \textbf{ontology} (combined complexity).
2. $\mathbf{AC^0}$ in the size of the \textbf{ABox} (data complexity).

**Proof (sketch).**

We observe that all the queries $q_N()$ and $q_F()$ checking for violations of negative inclusions $N$ and functionality assertions $F$ can be combined into a single UCQ whose size is linear in the TBox, and does not depend on the ABox. Hence, the result follows directly from the complexity of query answering over satisfiable ontologies.
Complexity of TBox reasoning

**Theorem**

TBox reasoning over $DL$-Lite$_A$ ontologies is PTIME in the size of the TBox (schema complexity).

**Proof (sketch).**

Follows from the previous theorem, and from the fact that all TBox reasoning tasks can be reduced to ontology satisfiability. Indeed, the size of the ontology constructed in the reduction is polynomial in the size of the input TBox.
Outline of Part 5

1. TBox reasoning
2. TBox & ABox reasoning and query answering
3. Beyond *DL-Lite*
   - Data complexity of query answering in DLs beyond *DL-Lite*
   - \textsc{NLogSpace}-hard DLs
   - \textsc{PTime}-hard DLs
   - \textsc{CoNP}-hard DLs
   - Combining functionality and role inclusions
   - Unique name assumption
Outline of Part 5

1. TBox reasoning

2. TBox & ABox reasoning and query answering

3. Beyond *DL-Lite*
   - Data complexity of query answering in DLs beyond *DL-Lite*
   - *NLOGSPACE*-hard DLs
   - *PTIME*-hard DLs
   - *coNP*-hard DLs
   - Combining functionality and role inclusions
   - Unique name assumption
Beyond *DL-Lite*

We consider now DL languages that extend *DL-Lite* with additional DL **constructs** or with combinations of constructs that are not legal in *DL-Lite*.

We show that (essentially) all such extensions of *DL-Lite* make it lose its nice computational properties.

Specifically, we consider the following DL constructs:

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>conjunction</td>
<td>$C_1 \sqcap C_2$</td>
<td><em>Doctor</em> $\sqcap$ <em>Male</em></td>
<td>$C_I^T \cap C_2^T$</td>
</tr>
<tr>
<td>disjunction</td>
<td>$C_1 \sqcup C_2$</td>
<td><em>Doctor</em> $\sqcup$ <em>Lawyer</em></td>
<td>$C_1^T \cup C_2^T$</td>
</tr>
<tr>
<td>qual. exist. restr.</td>
<td>$\exists Q.C$</td>
<td>$\exists child.Male$</td>
<td>{a</td>
</tr>
<tr>
<td>qual. univ. restr.</td>
<td>$\forall Q.C$</td>
<td>$\forall child.Male$</td>
<td>{a</td>
</tr>
</tbody>
</table>
## Beyond \( DL-Lite_A \): results on data complexity

<table>
<thead>
<tr>
<th>Lhs</th>
<th>Rhs</th>
<th>Funct.</th>
<th>Role incl.</th>
<th>Data complexity of query answering</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ( DL-Lite_A )</td>
<td>( A )</td>
<td>( \sqrt* )</td>
<td>( \sqrt* )</td>
<td>in ( AC^0 )</td>
</tr>
<tr>
<td>1</td>
<td>( A \upharpoonright \exists P.A )</td>
<td>( A )</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>( A )</td>
<td>( A \upharpoonright \forall P.A )</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>3</td>
<td>( A )</td>
<td>( A \upharpoonright \exists P.A )</td>
<td>( \sqrt )</td>
<td>–</td>
</tr>
<tr>
<td>4</td>
<td>( A \upharpoonright \exists P.A \upharpoonright A_1 \cap A_2 )</td>
<td>( A )</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>5</td>
<td>( A \upharpoonright A_1 \cap A_2 )</td>
<td>( A \upharpoonright \forall P.A )</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>6</td>
<td>( A \upharpoonright A_1 \cap A_2 )</td>
<td>( A \upharpoonright \exists P.A )</td>
<td>( \sqrt )</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>( A \upharpoonright \exists P.A \upharpoonright \exists P^- .A )</td>
<td>( A \upharpoonright \exists P )</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>( A \upharpoonright \exists P \upharpoonright \exists P^- )</td>
<td>( A \upharpoonright \exists P \upharpoonright \exists P^- )</td>
<td>( \sqrt )</td>
<td>( \sqrt )</td>
</tr>
<tr>
<td>9</td>
<td>( A \upharpoonright \neg A )</td>
<td>( A )</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>10</td>
<td>( A )</td>
<td>( A \upharpoonright A_1 \sqcup A_2 )</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>11</td>
<td>( A \upharpoonright \forall P.A )</td>
<td>( A )</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

### Notes:
- * with the “proviso” of not specializing functional properties.
- \( \text{NLogSpace} \) and \( \text{PTIME} \) hardness holds already for instance checking.
- For \( \text{coNP-hardness} \) in line 10, a TBox with a single assertion \( A_L \sqsubseteq A_T \sqcup A_F \) suffices! \( \sim \) No hope of including covering constraints.
Observations

- **DL-Lite-family** is FOL-rewritable, hence \( AC^0 \) – holds also with \( n \)-ary relations \( \leadsto DLR-Lite_\mathcal{F} \) and \( DLR-Lite_\mathcal{R} \).

- **RDFS** is a subset of \( DL-Lite_\mathcal{R} \) \( \leadsto \) is FOL-rewritable, hence \( AC^0 \).

- **Horn-SHIQ** [Hustadt et al. 2005] is \( PTIME\text{-}hard \) even for instance checking (line 8).

- **DLP** [Grosof et al. 2003] is \( PTIME\text{-}hard \) (line 4)

- **\( \mathcal{EL} \)** [Baader et al. 2005] is \( PTIME\text{-}hard \) (line 4).

- Although used in **ER** and **UML**, no hope of including **covering constraints**, since we get \( CONP\)-hardness for trivial DLs (line 10).
Outline of Part 5

1. TBox reasoning

2. TBox & ABox reasoning and query answering

3. Beyond DL-Lite
   - Data complexity of query answering in DLs beyond DL-Lite
   - NLogSpace-hard DLs
   - PTime-hard DLs
   - coNP-hard DLs
   - Combining functionality and role inclusions
   - Unique name assumption
Qualified existential quantification in the lhs of inclusions

Adding **qualified existential on the lhs** of inclusions makes instance checking (and hence query answering) $\text{NLogSpace}$-hard:

<table>
<thead>
<tr>
<th>Lhs</th>
<th>Rhs</th>
<th>$\mathcal{F}$</th>
<th>$\mathcal{R}$</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \mid \exists P.A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{NLogSpace}$-hard</td>
</tr>
</tbody>
</table>

Hardness proof is by a reduction from reachability in directed graphs:

- **ABox $\mathcal{A}$**: encodes graph using $P$ and asserts $A(d)$
- **TBox $\mathcal{T}$**: a single inclusion assertion $\exists P.A \sqsubseteq A$

Result:

$\langle \mathcal{T}, \mathcal{A} \rangle \models A(s)$ iff $d$ is reachable from $s$ in the graph.

**Note**: Since the reduction has to show hardness in data complexity, the graph must be encoded in the ABox (while the TBox has to be fixed).
Instance checking (and hence query answering) is $\text{NLogSpace}$-hard in data complexity for:

<table>
<thead>
<tr>
<th>Lhs</th>
<th>Rhs</th>
<th>$\mathcal{F}$</th>
<th>$\mathcal{R}$</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \mid \exists P . A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{NLogSpace}$-hard</td>
</tr>
</tbody>
</table>

By reduction from reachability in directed graphs.

2. $A \mid \forall P . A$  $A$  $-$  $-$  $\text{NLogSpace}$-hard

Follows from 1 by replacing $\exists P . A_1 \sqsubseteq A_2$ with $A_1 \sqsubseteq \forall P^- . A_2$, and by replacing each occurrence of $P^-$ with $P'$, for a new role $P'$.

3. $A \mid \exists P . A$  $A$  $\sqrt{\text{ }}$  $-$  $\text{NLogSpace}$-hard

Proved by simulating in the reduction $\exists P . A_1 \sqsubseteq A_2$ via $A_1 \sqsubseteq \exists P^- . A_2$ and ($\text{funct } P^-$), and by replacing again each occurrence of $P^-$ with $P'$, for a new role $P'$.
Outline of Part 5

1. TBox reasoning

2. TBox & ABox reasoning and query answering

3. Beyond DL-Lite
   - Data complexity of query answering in DLs beyond DL-Lite
   - NLOGSPACE-hard DLs
   - PTIME-hard DLs
   - coNP-hard DLs
   - Combining functionality and role inclusions
   - Unique name assumption
## PTIME-hard cases

<table>
<thead>
<tr>
<th>Lhs</th>
<th>Rhs</th>
<th>(\mathcal{F})</th>
<th>(\mathcal{R})</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (A \mid \exists P.A \mid A_1 \sqcap A_2)</td>
<td>(A)</td>
<td>–</td>
<td>–</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>5 (A \mid A_1 \sqcap A_2)</td>
<td>(A \mid \forall P.A)</td>
<td>–</td>
<td>–</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>6 (A \mid A_1 \sqcap A_2)</td>
<td>(A \mid \exists P.A)</td>
<td>√</td>
<td>–</td>
<td>PTIME-hard</td>
</tr>
</tbody>
</table>

The first three cases are obtained from the \(\text{NLogSpace}\)-hard cases by adding conjunction to the left-hand side of inclusions.

To show PTIME-hardness, we use a reduction similar to the one from reachability in directed graphs, but with a form of “non-linear” reachability.

<table>
<thead>
<tr>
<th>Lhs</th>
<th>Rhs</th>
<th>(\mathcal{F})</th>
<th>(\mathcal{R})</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (A \mid \exists P.A \mid \exists P^{-}.A)</td>
<td>(A \mid \exists P)</td>
<td>–</td>
<td>–</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>8 (A \mid \exists P \mid \exists P^{-})</td>
<td>(A \mid \exists P \mid \exists P^{-})</td>
<td>√</td>
<td>√</td>
<td>PTIME-hard</td>
</tr>
</tbody>
</table>

The remaining two cases require a different form of encoding.
Path System Accessibility

To show \textsc{PTime}-hardness, we use a reduction from a \textsc{PTime}-complete problem. We use Path System Accessibility.

Instance of Path System Accessibility: $PS = (N, E, S, t)$ with

- \(N\) a set of nodes
- \(E \subseteq N \times N \times N\) an accessibility relation
- \(S \subseteq N\) a set of source nodes
- \(t \in N\) a terminal node

\textbf{Accessibility} of nodes is defined inductively:

- each \(n \in S\) is accessible
- if \((n, n_1, n_2) \in E\) and \(n_1, n_2\) are accessible, then also \(n\) is accessible

Given an instance $PS$ of Path System Accessibility, deciding whether $t$ is accessible, is \textsc{PTime}-complete.
Reduction from Path System Accessibility

- Given an instance $PS = (N, E, S, t)$, we construct an ABox $A$ that:
  - encodes the accessibility relation using three roles $P_1$, $P_2$, and $P_3$, and
  - asserts $A(s)$ for each source node $s \in S$.

  $e_1 = (n, . , .)$
  $e_2 = (n, s_1, s_2)$
  $e_3 = (n, . , .)$

- We construct a TBox $\mathcal{T}$ consisting of the inclusion assertions:

  $\exists P_1 . A \sqsubseteq B_1$
  $\exists P_2 . A \sqsubseteq B_2$
  $B_1 \cap B_2 \sqsubseteq A$
  $\exists P_3 . A \sqsubseteq A$

Result:
$\langle \mathcal{T}, A \rangle \models A(t)$ iff $t$ is accessible in $PS$. 
Outline of Part 5

1. TBox reasoning
2. TBox & ABox reasoning and query answering
3. Beyond DL-Lite
   - Data complexity of query answering in DLs beyond DL-Lite
   - \textsc{NLogSpace}-hard DLs
   - \textsc{PTime}-hard DLs
   - \textsc{coNP}-hard DLs
   - Combining functionality and role inclusions
   - Unique name assumption
**coNP-hard cases**

Are obtained when we can use in the query **two concepts that cover another concept**. This forces **reasoning by cases** on the data.

Query answering is **coNP-hard** in data complexity for:

<table>
<thead>
<tr>
<th>Lhs</th>
<th>Rhs</th>
<th>(\mathcal{F})</th>
<th>(\mathcal{R})</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>(A \mid \neg A)</td>
<td>(A)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>10</td>
<td>(A)</td>
<td>(A \mid A_1 \sqcup A_2)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>11</td>
<td>(A \mid \forall P.A)</td>
<td>(A)</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

All three cases are proved by adapting the proof of coNP-hardness of instance checking for \(\mathcal{ALC}\) by [Donini et al. 1994].
2+2-SAT: satisfiability of a 2+2-CNF formula, i.e., a CNF formula where each clause has exactly 2 positive and 2 negative literals.

Example: $\varphi = c_1 \land c_2 \land c_3$, with

- $c_1 = v_1 \lor v_2 \lor \neg v_3 \lor \neg v_4$
- $c_2 = \text{false} \lor \text{false} \lor \neg v_1 \lor \neg v_4$
- $c_3 = \text{false} \lor v_4 \lor \text{true} \lor \neg v_2$

2+2-SAT is NP-complete [Donini et al. 1994].
Reduction from 2+2-SAT

We construct a TBox $\mathcal{T}$ and a query $q()$ over concepts $L$, $T$, $F$ and roles $P_1$, $P_2$, $N_1$, $N_2$.

- TBox $\mathcal{T} = \{ L \sqsubseteq T \sqcup F \} $
- $q() \leftarrow P_1(c, v_1), P_2(c, v_2), N_1(c, v_3), N_2(c, v_4), F(v_1), F(v_2), T(v_3), T(v_4)$

Given a 2+2-CNF formula $\varphi = c_1 \land \cdots \land c_k$ over vars $v_1, \ldots, v_n$, $true$, $false$, we construct an ABox $A_\varphi$ using individuals $c_1, \ldots c_k, v_1, \ldots, v_n$, $true$, $false$:

- for each propositional variable $v_i$: $L(v_i)$
- for each clause $c_j = v_{j_1} \lor v_{j_2} \lor \neg v_{j_3} \lor \neg v_{j_4}$:
  - $P_1(c_j, v_{j_1}), P_2(c_j, v_{j_2}), N_1(c_j, v_{j_3}), N_2(c_j, v_{j_4})$
- $T(true), F(false)$

Note: the TBox $\mathcal{T}$ and the query $q$ do not depend on $\varphi$, hence this reduction works for data complexity.
Reduction from $2+2$-SAT (cont’d)

Lemma

$\langle \mathcal{T}, A_\varphi \rangle \not\models q()$ iff $\varphi$ is satisfiable.

Proof (sketch).

“$\Rightarrow$” If $\langle \mathcal{T}, A_\varphi \rangle \not\models q()$, then there is a model $\mathcal{I}$ of $\langle \mathcal{T}, A_\varphi \rangle$ s.t. $\mathcal{I} \not\models q()$. We define a truth assignment $\alpha_\mathcal{I}$ by setting $\alpha_\mathcal{I}(v_i) = true$ iff $v^\mathcal{I}_i \in T^\mathcal{I}$. Notice that, since $L \subseteq T \sqcup F$, if $v^\mathcal{I}_i \notin T^\mathcal{I}$, then $v^\mathcal{I}_i \in F^\mathcal{I}$.

It is easy to see that, since $q()$ asks for a false clause and $\mathcal{I} \not\models q()$, for each clause $c_j$, one of the literals in $c_j$ evaluates to $true$ in $\alpha_\mathcal{I}$.

“$\Leftarrow$” From a truth assignment $\alpha$ that satisfies $\varphi$, we construct an interpretation $\mathcal{I}_\alpha$ with $\Delta^{\mathcal{I}_\alpha} = \{c_1, \ldots, c_k, v_1, \ldots, v_n, t, f\}$, and:

- $c_j^{\mathcal{I}_\alpha} = c_j$, $v_i^{\mathcal{I}_\alpha} = v_i$, $true^{\mathcal{I}_\alpha} = t$, $false^{\mathcal{I}_\alpha} = f$
- $T^{\mathcal{I}_\alpha} = \{v_i \mid \alpha(v_i) = true\} \cup \{t\}$, $F^{\mathcal{I}_\alpha} = \{v_i \mid \alpha(v_i) = false\} \cup \{f\}$

It is easy to see that $\mathcal{I}_\alpha$ is a model of $\langle \mathcal{T}, A_\varphi \rangle$ and that $\mathcal{I}_\alpha \not\models q()$. 


Outline of Part 5

1. TBox reasoning
2. TBox & ABox reasoning and query answering
3. Beyond DL-Lite
   - Data complexity of query answering in DLs beyond DL-Lite
   - NLOGSPACE-hard DLs
   - PTIME-hard DLs
   - coNP-hard DLs
   - Combining functionality and role inclusions
   - Unique name assumption
Combining functionalities and role inclusions

Let $DL-Lite_{\mathcal{FR}}$ be the DL that is the union of $DL-Lite_{\mathcal{F}}$ and $DL-Lite_{\mathcal{R}}$, i.e., the $DL-Lite$ logic that allows for using both role functionality and role inclusions without any restrictions.

Due to the unrestricted interaction of functionality and role inclusions $DL-Lite_{\mathcal{FR}}$ is significantly more complicated than the logics of the $DL-Lite$ family:

- One can force the unification of existentially implied objects (i.e., separation does not hold anymore).
- Additional constructs besides those present in $DL-Lite$ can be simulated.
- The computational complexity of reasoning increases significantly.
Unification of existentially implied objects – Example

TBox $\mathcal{T}$:

- $A \sqsubseteq \exists P$
- $P \sqsubseteq S$
- $\exists P^- \sqsubseteq A$ (funct $S$)

ABox $\mathcal{A}$:

- $A(c_1)$, $S(c_1, c_2)$, $S(c_2, c_3)$, \ldots, $S(c_{n-1}, c_n)$
- $A(c_1)$, $A \sqsubseteq \exists P$ $\models$ $P(c_1, x)$, for some $x$
- $P(c_1, x)$, $P \sqsubseteq S$ $\models$ $S(c_1, x)$
- $S(c_1, x)$, $S(c_1, c_2)$, (funct $S$) $\models$ $x = c_2$
- $P(c_1, c_2)$, $\exists P^- \sqsubseteq A$ $\models$ $A(c_2)$
- $A(c_2)$, $A \sqsubseteq \exists P$ $\models$ $A(c_n)$

Hence, we get:

- If we add $B(c_n)$ and $B \sqsubseteq \neg A$, the ontology becomes inconsistent.
- Similarly, the answer to the following query over $\langle \mathcal{T}, \mathcal{A} \rangle$ is true:

$$q() \leftarrow A(z_1), S(z_1, z_2), S(z_2, z_3), \ldots, S(z_{n-1}, z_n), A(z_n)$$
Unification of existentially implied objects

Note: The number of unification steps above depends on the data. Hence this kind of deduction cannot be mimicked by a FOL (or SQL) query, since it requires a form of recursion. As a consequence, we get:

Combining functionality and role inclusions is problematic.

It breaks separability, i.e., functionality assertions may force existentially quantified objects to be unified with existing objects.

Note: the problems are caused by the interaction among:
- an inclusion $P \sqsubseteq S$ between roles,
- a functionality assertion (funct $S$) on the super-role, and
- a cycle of concept inclusion assertions $A \sqsubseteq \exists P$ and $\exists P^- \sqsubseteq A$. 
In fact, by exploiting the interaction between functionality and role inclusions, we can simulate typical DL constructs not present in \textit{DL-Lite}:

- **Simulation of** $A \sqsubseteq \exists R.C$: \textit{(Note:} this does not require functionality\textit{)}

  \[
  A \sqsubseteq \exists R_C \quad R_C \sqsubseteq R \quad \exists R_C^- \sqsubseteq C
  \]

- **Simulation of** $A_1 \sqcap A_2 \sqsubseteq C$:

  \[
  A_1 \sqsubseteq \exists R_1 \quad A_2 \sqsubseteq \exists R_2 \\
  R_1 \sqsubseteq R_{12} \quad R_2 \sqsubseteq R_{12} \quad \text{(funct } R_{12}) \\
  \exists R_1^- \sqsubseteq \exists R_3^- \\
  \exists R_3 \sqsubseteq C \\
  R_3 \sqsubseteq R_{23} \quad R_2 \sqsubseteq R_{23} \quad \text{(funct } R_{23}^-)
  \]
Simulation of constructs (cont’d)

Simulation of $A \sqsubseteq \forall R.C$:

We use reification of roles:

$S_1, C \sqsubseteq S_1$

$S_2, C \sqsubseteq S_2$

$\exists S_1, C \equiv \exists S_2, C$

$\exists S_1, \neg C \equiv \exists S_2, \neg C$

$\exists S_2 \sqsubseteq \exists S_2, C \sqcup \exists S_2, \neg C$

$\exists S_2, C \sqsubseteq C$

$\exists S_2, \neg C \sqsubseteq \neg C$

$A \sqsubseteq \neg \exists S_1, \neg C$

(funct $S_1$)

(funct $S_2$)
We can exploit the above constructions that simulate DL constructs to show lower bounds for reasoning with both functionality and role inclusions.

**Theorem [Artale et al. 2009]**

For $DL-Lite_{\mathcal{FR}}$ ontologies:

- **TBox reasoning** is $\text{ExpTime-complete}$ in the size of the TBox.
- Checking **satisfiability of the ontology** is
  - $\text{PTime-complete}$ in the size of the ABox (data complexity).
  - $\text{ExpTime-complete}$ in the size of the ontology (combined complexity).
- **Query answering** is
  - $\text{PTime-complete}$ in the size of the ABox (data complexity).
  - $\text{ExpTime-complete}$ in the size of the ontology.
  - in $2\text{ExpTime}$ in the size of the query and the ontology (combined com.).
We have seen that:

- By including in \textit{DL-Lite} both functionality of roles and role inclusions without restrictions on their interaction, query answering becomes \texttt{PTime}-hard.
- When the data complexity of query answering is \texttt{NLogSpace} or above, the DL does not enjoy FOL-rewritability.

As a consequence of these results, we get:

To preserve FOL-rewritability, the restriction on the interaction of functionality and role inclusions of \textit{DL-Lite}_\textsubscript{A} is necessary.
Outline of Part 5

1. TBox reasoning

2. TBox & ABox reasoning and query answering

3. Beyond $DL$-$Lite$
   - Data complexity of query answering in DLs beyond $DL$-$Lite$
   - $NLOGSPACE$-hard DLs
   - $PTIME$-hard DLs
   - $coNP$-hard DLs
   - Combining functionality and role inclusions
   - Unique name assumption
Dropping the unique name assumption

*Recall:* the unique name assumption (UNA) states that different individuals must be interpreted as different domain objects.

We reconsider the complexity of query evaluation in $DL$-Lite$_F$, and show that **without the UNA the data complexity increases.**

- We show how to reduce *reachability in directed graphs* to instance checking in $DL$-Lite$_F$ without the UNA. This gives us an $\mathsf{NLogSpace}$ lower bound.
- We assume that the graph is represented through the first-child and next-sibling functional relations:
Dropping the unique name assumption (cont’d)

From $G$ and two vertexes $s$ and $t$ of $G$, we define $\mathcal{O}_{una} = \langle \mathcal{T}_{una}, \mathcal{A}_G \rangle$:

- TBox uses an atomic concept $A$, and atomic roles $P_0$, $P_F$, $P_N$, $P_S$:
  
  $$\mathcal{T}_{una} = \{ \text{funct } P_0 \} \cup \{ \text{funct } P_R \mid R \in \{ F, N, S \} \}.$$  

- ABox is defined from $G$ and the two vertexes $s$ and $t$:
  
  $$\mathcal{A}_G = \{ P_R(a_1, a_2), P_R(a'_1, a'_2) \mid (a_1, a_2) \in R, \text{ for } R \in \{ F, N, S \} \} \cup \{ A(t), P_0(a_{init}, s), P_0(a_{init}, s') \}.$$  

This means that we encode in $\mathcal{A}_G$ two copies of $G$.

Note: $\mathcal{A}_G$ depends on $G$, but $\mathcal{T}_{una}$ does not.

We can show by induction on the length of paths from $s$ that . . .

$t$ is reachable from $s$ in $G$ if and only if $\mathcal{O}_{una} \models A(t')$. 
The previous reduction shows that instance checking in \( DL-Lite_F \) (and hence also \( DL-Lite_A \)) without the UNA is \text{NLogSpace}-hard.

With a more involved reduction, one can show an even stronger lower bound, that turns out to be tight.

\textbf{Theorem} [Artale et al. 2009]

Instance checking in \( DL-Lite_F \) and \( DL-Lite_A \) without the UNA is \text{PTime}-complete in data complexity.
### References I


