Query Rewriting in OBDA
Forward Chaining Techniques

Useful techniques for establishing optimal upper bounds

…but not practical - we need to store instances of very large size
How we achieve true scalability in OBQA?
Scalability in OBQA

Exploit standard RDBMSs - efficient technology for answering CQs

But in the OBQA setting we have to query a knowledge base, not just a relational database
Query Rewriting

\[ \forall D: D \land \Sigma \models Q \iff D \models Q_\Sigma \]

First-order query
Union of CQs
SQL query
Datalog query
...

evaluated and optimized by exploiting existing technology
Consider a class of existential rules $L$, and a query language $Q$.

$\text{OBQA}(L)$ is $Q$-rewritable if, for every $\Sigma \in L$ and (Boolean) CQ $Q$,

we can construct a query $Q_\Sigma \in Q$ such that,

for every database $D$, $D \land \Sigma \models Q$ iff $D \models Q_\Sigma$

\textbf{NOTE:} The construction of $Q_\Sigma$ is database-independent - the pure approach to query rewriting
Issues in Query Rewriting

• How do we choose the target query language?

• How the ontology language and the target query language are related?

• How we construct such rewritings?

• What about the size of such rewritings?
Target Query Language

we target the weakest query language

<table>
<thead>
<tr>
<th></th>
<th>CQ</th>
<th>UCQ</th>
<th>FO</th>
<th>Datalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>ACYCLIC</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINEAR</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Target Query Language

we target the weakest query language

<table>
<thead>
<tr>
<th></th>
<th>CQ</th>
<th>UCQ</th>
<th>FO</th>
<th>Datalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>ACYCLIC</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINEAR</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Target Query Language

\[ \Sigma = \{ \forall x \ (P(x) \rightarrow T(x)), \ \forall x \forall y \ (R(x,y) \rightarrow S(x)) \} \]

\[ Q :- S(x), U(x,y), T(y) \]

\[ Q_\Sigma = \{ Q :- S(x), U(x,y), T(y), \\
Q_1 :- S(x), U(x,y), P(y), \\
Q_2 :- R(x,z), U(x,y), T(y), \\
Q_3 :- R(x,z), U(x,y), P(y) \} \]
Target Query Language

we target the weakest query language

<table>
<thead>
<tr>
<th></th>
<th>CQ</th>
<th>UCQ</th>
<th>FO</th>
<th>Datalog</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FULL</strong></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>ACYCLIC</strong></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>LINEAR</strong></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Target Query Language

\[ \Sigma = \{ \forall x \forall y (R(x,y) \land P(y) \rightarrow P(x)) \} \]

\[ Q :\!:= P(c) \]

\[ Q_\Sigma = \{ Q :\!:= P(c), \]

\[ Q_1 :\!:= R(c,y_1), P(y_1), \]

\[ Q_2 :\!:= R(c,y_1), R(y_1,y_2), P(y_2), \]

\[ Q_3 :\!:= R(c,y_1), R(y_1,y_2), R(y_2,y_3), P(y_3), \]

\[ \ldots \} \]

- This cannot be written as a finite UCQ (or even FO query)
- It can be written as \( Q :\!:= R(c,x), R^*(x,y), P(y) \), but transitive closure is not FO-expressible
Target Query Language

we target the weakest query language

<table>
<thead>
<tr>
<th></th>
<th>CQ</th>
<th>UCQ</th>
<th>FO</th>
<th>Datalog</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FULL</strong></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>ACYCLIC</strong></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>LINEAR</strong></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
UCQ-Rewritings

• The standard algorithm for computing UCQ-rewritings performs an exhaustive application of the following two steps:
  1. Rewriting
  2. Minimization

• The standard algorithm is designed for normalized existential rules, where only one atom appears in the head
Normalization Procedure

\[ \forall x \forall y (\varphi(x, y) \rightarrow \exists z (P_1(x, z) \land \ldots \land P_n(x, z))) \]

\[ \forall x \forall y (\varphi(x, y) \rightarrow \exists z \text{ Auxiliary}(x, z)) \]

\[ \forall x \forall z (\text{Auxiliary}(x, z) \rightarrow P_1(x, z)) \]

\[ \forall x \forall z (\text{Auxiliary}(x, z) \rightarrow P_2(x, z)) \]

\[ \ldots \]

\[ \forall x \forall z (\text{Auxiliary}(x, z) \rightarrow P_n(x, z)) \]

**NOTE 1:** Acyclicity and Linearity are preserved

**NOTE 2:** We obtain an equivalent set w.r.t. query answering
UCQ-Rewritings

• The standard algorithm for computing UCQ-rewritings performs an exhaustive application of the following two steps:
  1. Rewriting
  2. Minimization

• The standard algorithm is designed for normalized existential rules, where only one atom appears in the head
Rewriting Step

\[ \Sigma = \{ \forall x \forall y \ (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z \ \text{hasCollaborator}(z,y,x)) \} \]

\[ Q :- \ \text{hasCollaborator}(u,db,v) \]

\[ g = \{ x \rightarrow v, y \rightarrow db, z \rightarrow u \} \]

Thus, we can simulate a chase step by applying a backward resolution step

\[ Q_\Sigma = \{ Q :- \ \text{hasCollaborator}(u,db,v), Q_1 :- \ \text{project}(v), \ \text{inArea}(v,db) \} \]
Unsound Rewritings

\[ \Sigma = \{ \forall x \forall y \ (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z \ \text{hasCollaborator}(z,y,x)) \} \]

\[ Q : - \ \text{hasCollaborator}(c,db,v) \]

\[ g = \{ x \mapsto v, \ y \mapsto db, \ z \mapsto c \} \]

After applying the rewriting step we obtain the following UCQ

\[ Q_\Sigma = \{ Q : - \ \text{hasCollaborator}(c,db,v), \]

\[ Q_1 : - \ \text{project}(v), \ \text{inArea}(v,db) \} \]
Unsound Rewritings

\[ \Sigma = \{ \forall x \forall y (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z \text{ hasCollaborator}(z,y,x)) \} \]

\[ Q : - \text{hasCollaborator}(c,db,v) \]

\[ Q_\Sigma = \{ Q : - \text{hasCollaborator}(c,db,v), \\
\quad Q_1 : - \text{project}(v), \text{inArea}(v,db) \} \]

- Consider the database \( D = \{ \text{project}(a), \text{inArea}(a,db) \} \)

- Clearly, \( D \models Q_\Sigma \)

- However, \( D \land \Sigma \) does not entail \( Q \) since there is no way to obtain an atom of the form \( \text{hasCollaborator}(c,db,_) \) during the chase
Unsound Rewritings

Σ = {∀x∀y (project(x) ∧ inArea(x,y) → ∃z hasCollaborator(z,y,x))}

Q :- hasCollaborator(c,db,v)

\[ Q_Σ = \{ Q :- \text{hasCollaborator}(c,db,v), \\
          Q_1 :- \text{project}(v), \text{inArea}(v,db) \} \]

the information about the constant c in the original query is lost after the application of the rewriting step since c is unified with an ∃-variable
Unsound Rewritings

\[ \Sigma = \{ \forall x \forall y (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z \ \text{hasCollaborator}(z,y,x)) \} \]

\[ Q :- \ \text{hasCollaborator}(v,db,v) \]

\[ g = \{ x \rightarrow v, y \rightarrow db, z \rightarrow v \} \]

\[ \text{hasCollaborator}(v,db,v) \]

After applying the rewriting step we obtain the following UCQ

\[ Q_\Sigma = \{ Q :- \text{hasCollaborator}(v,db,v), Q_1 :- \text{project}(v), \text{inArea}(v,db) \} \]
Unsound Rewritings

\[ \Sigma = \{ \forall x \forall y (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z \text{ hasCollaborator}(z,y,x)) \} \]

\[ Q :\neg \text{ hasCollaborator}(v,db,v) \]

\[ Q_\Sigma = \{ Q :\neg \text{ hasCollaborator}(v,db,v), \\
Q_1 :\text{ project}(v), \text{ inArea}(v,db) \} \]

• Consider the database \( D = \{ \text{project}(a), \text{inArea}(a,db) \} \)

• Clearly, \( D \models Q_\Sigma \)

• However, \( D \land \Sigma \) does not entail \( Q \) since there is no way to obtain an atom of the form \( \text{hasCollaborator}(t,db,t) \) during the chase
Unsound Rewritings

\[ \Sigma = \{ \forall x \forall y (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z \ \text{hasCollaborator}(z,y,x)) \} \]

\[ Q :\neg \ \text{hasCollaborator}(v,\text{db},v) \]

\[ Q_\Sigma = \{ Q :\neg \ \text{hasCollaborator}(v,\text{db},v), \]

\[ Q_1 :\neg \ \text{project}(v), \ \text{inArea}(v,\text{db}) \} \]

the fact that \( v \) in the original query participates in a join is lost after the application of the rewriting step since \( v \) is unified with an \( \exists \)-variable
Applicability Condition

Consider a (Boolean) CQ $Q$, an atom $\alpha$ in $Q$, and a (normalized) rule $\sigma$.

We say that $\sigma$ is applicable to $\alpha$ if the following conditions hold:

1. head($\sigma$) and $\alpha$ unify via $h$

2. For every variable $x$ in head($\sigma$):
   1. If $h(x)$ is a constant, then $x$ is a $\forall$-variable
   2. If $h(x) = h(y)$, where $y$ is a shared variable of $\alpha$, then $x$ is a $\forall$-variable

3. If $x$ is an $\exists$-variable of head($\sigma$), and $y$ is a variable in head($\sigma$) such that $x \neq y$, then $h(x) \neq h(y)$

...but, although is crucial for soundness, may destroy completeness
Incomplete Rewritings

\[ \Sigma = \{ \forall x \forall y (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z \ \text{hasCollaborator}(z,y,x)), \]
\[ \forall x \forall y \forall z (\text{hasCollaborator}(x,y,z) \rightarrow \text{collaborator}(x)) \}\]

\[ Q :\neg \text{hasCollaborator}(u,v,w), \text{collaborator}(u) \]

\[ Q_\Sigma = \{ Q :\neg \text{hasCollaborator}(u,v,w), \text{collaborator}(u), \]
\[ Q_1 :\neg \text{hasCollaborator}(u,v,w), \text{hasCollaborator}(u,v',w') \]

- Consider the database \( D = \{ \text{project}(a), \text{inArea}(a,db) \} \)
- Clearly, \( \text{chase}(D, \Sigma) = D \cup \{ \text{hasCollaborator}(z,db,a), \text{collaborator}(z) \} \models Q \)
- However, \( D \) does not entail \( Q_\Sigma \)
Incomplete Rewritings

\[ \Sigma = \{ \forall x \forall y \ (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z \ \text{hasCollaborator}(z,y,x)), \]
\[ \forall x \forall y \forall z \ (\text{hasCollaborator}(x,y,z) \rightarrow \text{collaborator}(x)) \}\]

\[ Q :\neg \ \text{hasCollaborator}(u,v,w), \ \text{collaborator}(u) \]

\[ Q_\Sigma = \{ Q :\neg \ \text{hasCollaborator}(u,v,w), \ \text{collaborator}(u), \]
\[ Q_1 :\neg \ \text{hasCollaborator}(u,v,w), \ \text{hasCollaborator}(u,v',w') \]
\[ Q_2 :\neg \ \text{project}(u), \ \text{inArea}(u,v) \]

...but, we cannot obtain the last query due to the applicability condition
Incomplete Rewritings

\[ \Sigma = \{ \forall x \forall y (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z \, \text{hasCollaborator}(z,y,x)), \]
\[ \forall x \forall y \forall z (\text{hasCollaborator}(x,y,z) \rightarrow \text{collaborator}(x)) \} \]

\[ Q \maps \text{hasCollaborator}(u,v,w), \text{collaborator}(u) \]

\[
Q_\Sigma = \{ Q \maps \text{hasCollaborator}(u,v,w), \text{collaborator}(u), \\
Q_1 \maps \text{hasCollaborator}(u,v,w), \text{hasCollaborator}(u,v',w') \\
Q_2 \maps \text{hasCollaborator}(u,v,w) - \text{by minimization} \\
Q_3 \maps \text{project}(w), \text{inArea}(w,v) - \text{by rewriting} \}
\]

\[ D = \{ \text{project}(a), \text{inArea}(a,db) \} \models Q_\Sigma \]
UCQ-Rewritings

• The standard algorithm for computing UCQ-rewritings performs an exhaustive application of the following two steps:
  1. Rewriting
  2. Minimization

• The standard algorithm is designed for normalized existential rules, where only one atom appears in the head
The Rewriting Algorithm

\[ Q_\Sigma := \{Q\}; \]
repeat
\[ Q_{aux} := Q_\Sigma; \]
\[ \textbf{foreach} \text{ disjunct } q \text{ of } Q_{aux} \text{ do} \]
\[ //\text{Rewriting Step} \]
\[ \textbf{foreach} \text{ atom } \alpha \text{ in } q \text{ do} \]
\[ \textbf{foreach} \text{ rule } \sigma \text{ in } \Sigma \text{ do} \]
\[ \text{if } \sigma \text{ is applicable to } \alpha \text{ then} \]
\[ q_{rew} := \text{rewrite}(q, \alpha, \sigma); \quad //\text{we resolve } \alpha \text{ using } \sigma \]
\[ \text{if } q_{rew} \text{ does not appear in } Q_\Sigma \text{ (modulo variable renaming) then} \]
\[ Q_\Sigma := Q_\Sigma \cup \{q_{rew}\}; \]
\[ //\text{Minimization Step} \]
\[ \textbf{foreach} \text{ pair of atoms } \alpha, \beta \text{ in } q \text{ that unify do} \]
\[ q_{min} := \text{minimize}(q, \alpha, \beta); \quad //\text{we apply the MGU of } \alpha \text{ and } \beta \text{ on } q \]
\[ \text{if } q_{min} \text{ does not appear in } Q_\Sigma \text{ (modulo variable renaming) then} \]
\[ Q_\Sigma := Q_\Sigma \cup \{q_{min}\}; \]
until \[ Q_{aux} = Q_\Sigma; \]
return \[ Q_\Sigma; \]
Termination

**Theorem:** The rewriting algorithm terminates under **ACYCLIC**

**Proof Idea:**

- **Key observation:** after arranging the disjuncts of the rewriting in a tree $T$, the branching of $T$ is finite, and the depth of $T$ is at most the number of predicates occurring in the rule set

- Therefore, only finitely many partial rewritings can be constructed - in general, exponentially many
Termination

**Theorem:** The rewriting algorithm terminates under **LINEAR**

**Proof Idea:**

- **Key observation:** the size of each partial rewriting is at most the size of the given CQ $Q$

- Thus, each partial rewriting can be transformed into an equivalent query that contains at most $(|Q| \cdot \text{maxarity})$ variables

- The number of queries that can be constructed using a finite number of predicates and a finite number of variables is finite

- Therefore, only finitely many partial rewritings can be constructed - in general, exponentially many
Target Query Language

we target the weakest query language

```
<table>
<thead>
<tr>
<th></th>
<th>CQ</th>
<th>UCQ</th>
<th>FO</th>
<th>Datalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>ACYCLIC</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINEAR</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
```
### Data Complexity

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Data Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>PTIME-(c)</td>
<td>Naïve algorithm</td>
</tr>
<tr>
<td></td>
<td>Reduction from Monotone Circuit Value problem</td>
<td></td>
</tr>
<tr>
<td>ACYCLIC</td>
<td>in LOGSPACE</td>
<td>Via UCQ-rewriting</td>
</tr>
<tr>
<td>LINEAR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Combined Complexity

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Combined Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>EXPTIME-(c)</td>
<td>Naïve algorithm</td>
</tr>
<tr>
<td></td>
<td>Simulation of a deterministic exponential time TM</td>
<td></td>
</tr>
<tr>
<td>ACYCLIC</td>
<td>NEXPTIME-(c)</td>
<td>Small witness property</td>
</tr>
<tr>
<td></td>
<td>Reduction from a Tiling problem</td>
<td></td>
</tr>
<tr>
<td>LINEAR</td>
<td>PSPACE-(c)</td>
<td>Level-by-level non-deterministic algorithm</td>
</tr>
<tr>
<td></td>
<td>Simulation of a deterministic polynomial space TM</td>
<td></td>
</tr>
</tbody>
</table>
Size of the Rewriting

• Ideally, we would like to construct UCQ-rewritings of polynomial size

• But, the standard rewriting algorithm produces rewritings of exponential size

• Can we do better? NO!!!

\[ \Sigma = \{ \forall x \ (R_k(x) \rightarrow P_k(x)) \}_{k \in \{1,\ldots,n\}} \]

\[ Q ::= P_1(x), \ldots, P_n(x) \]

\[ Q ::= P_1(X), \ldots, P_n(X) \]

\[ P_1(X) \lor R_1(X) \]

\[ P_n(X) \lor R_n(X) \]

thus, we need to consider \(2^n\) disjuncts
Size of the Rewriting

• Ideally, we would like to construct UCQ-rewritings of polynomial size

• But, the standard rewriting algorithm produces rewritings of exponential size

• Can we do better? **NO!!!**

• Although the standard rewriting algorithm is worst-case optimal, it can be significantly improved

• Optimization techniques can be applied in order to compute efficiently small rewritings - field of intense research
Limitations of UCQ-Rewritability

∀D : D ∧ Σ ⊨ Q ⇔ D ⊨ Q_Σ

evaluated and optimized by exploiting existing technology

• What about the size of Q_Σ? - very large, no rewritings of polynomial size

• What kind of ontology languages can be used for Σ? - below PTIME

⇒ the combined approach to query rewriting
Combined Rewritability

\[ \forall D : D \land \Sigma \models Q \iff D^+ \models Q_\Sigma \]
Polynomial Combined Rewritability

\[ \forall D : D \land \Sigma \models Q \iff D^+ \models Q_\Sigma \]