Query Answering under Existing Rules
An Overview

Sebastian Rudolph
TU Dresden

ICCL Summer School 2015
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
- KR View: KB Rewriting into Nice Models
  - Finite models through Acyclicity
  - Bounded-treewidth Models through Guardedness
  - Joining Acyclicity and Guardedness
  - Algorithmic Aspects
- DB View: Query rewriting
- Mixing the Views
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
- KR View: KB Rewriting into Nice Models
  - Finite models through Acyclicity
  - Bounded-treewidth Models through Guardedness
  - Joining Acyclicity and Guardedness
  - Algorithmic Aspects
- DB View: Query rewriting
- Mixing the Views
Classical Query Answering

Data/Facts ➔ Answers ? ➔ Query
### Data / Facts

#### Relational Database

<table>
<thead>
<tr>
<th>parentOf</th>
<th>Male</th>
<th>Fem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

| A | B | A | C | X | ... |

#### RDF (Semantic Web)

- F
- M
- ex:A
- ex:B
- ex:C

\[ \exists x ( \text{parentOf}(A,B) \land \text{parentOf}(A,C) \land \text{parentOf}(C,x) \land F(A) \land M(B) \land M(x) ) \]

**Abstraction in First-Order Logic**

**Or in graphs / hypergraphs**
Queries

Typically expressed as formulae of some logic (the query language) with free variables.

A lot of options, tradeoff between expressivity and computational well-behavedness.
Conjunctive Queries

Example:

« Find all x such that x is a female and has a child who is a female »

\[ \exists y \ (\text{Female}(x) \land \text{childOf}(x, y) \land \text{Female}(y)) \]

FOL formula

\[ Q(x) = \text{Female}(x), \text{childOf}(x, y), \text{Female}(y) \]

Common notation

\[ \text{ans}(x) \leftarrow \text{Female}(x), \text{childOf}(x, y), \text{Female}(y) \]

Datalog notation

SELECT ... FROM ... WHERE ...

SQL/SPARQL

Formally: A CQ \( Q \) has the form \( \exists x_{k+1},...,x_m \ A_1 \land ... \land A_p \)

where \( A_1,...,A_p \) are atoms over the variables \( x_1,...,x_m \)

and \( x_1 \ldots x_k \) are free variables (defining the answer part)

If \( k = 0 \), \( Q \) is a Boolean CQ (existentially closed conjunctive formula)

then the answer can only be yes or no.
Evaluating Boolean CQs over Data

Data:

\[ \exists x \ (\text{loves}(bob,bob) \land \text{hates}(bob,x) \land \text{hates}(alice,bob)) \]

Query:

\[ \exists xyz \ (\text{loves}(x,y) \land \text{hates}(x,z) \land \text{hates}(y,z)) \]

Homomorphism
Some well-known query languages

- Conjunctive queries (CQ)
  - ... and their unions
- First-order logic (FOL)
  - Basis of SQL
- Datalog
  - Recursive, higher-order language
- Second-order logic (SO)
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
- KR View: KB Rewriting into Nice Models
  - Finite models through Acyclicity
  - Bounded-treewidth Models through Guardedness
  - Joining Acyclicity and Guardedness
  - Algorithmic Aspects
- DB View: Query rewriting
- Mixing the Views
Why Ontological Query Answering?

- vocabulary of data and query may not coincide (→ information exchange)
- databases may be incomplete
- some information may only be obtained when factoring in background knowledge
The Plain View:

\[ D \models_{\Sigma} Q \]
The Knowledge Representation View:

\[ D \land \Sigma \models Q \]
The Database View:

$$D \models (\land \Sigma) \Rightarrow Q$$
Our Main Showcase:

Data  Ontology  Query
Our Main Showcase:

Data → Existential Rules → Boolean Conjunctive Query
Existential Rules

\[ \forall X \forall Y ( B[X, Y] \rightarrow \exists Z H[X, Z] ) \]

\( X, Y, Z : \) tuples of variables

Any conjunction of atoms (on variables and constants)

\[ \forall x \forall y ( \text{ siblingOf}(x,y) \rightarrow \exists z ( \text{ parentOf}(z,x) \land \text{ parentOf}(z,y))) \]

Simplified form: \( \text{ siblingOf}(x,y) \rightarrow \text{ parentOf}(z,x) \land \text{ parentOf}(z,y) \)

- Same as Tuple Generating Dependencies (TGDs)
- See also Datalog+/- [Cali Gottlob Lukasiewicz PODS 2009]
- Same as the logical translation of Conceptual Graph rules
- Generalize lightweight DLs used for OBQA
## DL-Lite Family

**DL-Lite**: Popular family of DLs - at the basis of the OWL 2 QL profile of OWL

<table>
<thead>
<tr>
<th>DL-Lite Axioms</th>
<th>First-order Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \subseteq B$</td>
<td>$\forall x \ (A(x) \rightarrow B(x))$</td>
</tr>
<tr>
<td>$A \subseteq \exists R$</td>
<td>$\forall x \ (A(x) \rightarrow \exists y \ R(x,y))$</td>
</tr>
<tr>
<td>$\exists R \subseteq A$</td>
<td>$\forall x \forall y \ (R(x,y) \rightarrow A(x))$</td>
</tr>
<tr>
<td>$\exists R \subseteq \exists P$</td>
<td>$\forall x \forall y \ (R(x,y) \rightarrow \exists z \ P(x,z))$</td>
</tr>
<tr>
<td>$A \subseteq \exists R . B$</td>
<td>$\forall x \ (A(x) \rightarrow \exists y \ (R(x,y) \land B(y)))$</td>
</tr>
<tr>
<td>$R \subseteq P$</td>
<td>$\forall x \forall y \ (R(x,y) \rightarrow P(x,y))$</td>
</tr>
<tr>
<td>$A \subseteq \neg B$</td>
<td>$\forall x \ (A(x) \land B(x) \rightarrow \bot)$</td>
</tr>
</tbody>
</table>
The Description Logic EL

EL: Popular DL for biological applications - at the basis of OWL 2 EL profile

<table>
<thead>
<tr>
<th>EL Axioms</th>
<th>First-order Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \sqsubseteq B$</td>
<td>$\forall x \ (A(x) \rightarrow B(x))$</td>
</tr>
<tr>
<td>$A \cap B \sqsubseteq C$</td>
<td>$\forall x \ (A(x) \land B(x) \rightarrow C(x))$</td>
</tr>
<tr>
<td>$A \sqsubseteq \exists R.B$</td>
<td>$\forall x \ (A(x) \rightarrow \exists y \ (R(x,y) \land B(y)))$</td>
</tr>
<tr>
<td>$\exists R.B \sqsubseteq A$</td>
<td>$\forall x \forall y \ (R(x,y) \land B(y) \rightarrow A(x))$</td>
</tr>
</tbody>
</table>
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
  - KR View: KB Rewriting into Nice Models
    - Finite models through Acyclicity
    - Bounded-treewidth Models through Guardedness
    - Joining Acyclicity and Guardedness
    - Algorithmic Aspects
  - DB View: Query rewriting
- Mixing the Views
rewrite knowledge base into a representation that is easier to query, by making the structure of its models more explicit

- for FOL: semantic tableau
  (also used for Description Logics)
- for Existential Rules: the chase
The Chase

\[ \Sigma \]

\[ t(x) \rightarrow s(x, y) \land t(y) \]

\[ t(x) \land t(y) \rightarrow r(x, y) \]

\[ D = \{ t(A) \} \]

\[ \text{chase}(D, \Sigma) \]

Diagram showing the chase process with nodes and edges representing the rules and data.
The Chase

- $D \models_{\Sigma} Q$ iff chase$(D, \Sigma)$ satisfies $Q$
- works for all query languages where satisfaction is preserved under homomorphism (not just for CQs)
- produces a universal model
The Chase

- \( D \models_{\Sigma} Q \) iff \( \text{chase}(D,\Sigma) \) satisfies \( Q \)
- works for all query languages where satisfaction is preserved under homomorphism (not just for CQs)
- produces a universal model
- but: not sufficient for decidability, only in cases where the chase is known to have further properties:
  - finiteness
  - bounded tree-width
- checking these properties is again undecidable
- there exist sufficient (groups of) criteria
  - acyclicity
  - guardedness
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
- KR View: KB Rewriting into Nice Models
  - Finite models through Acyclicity
  - Bounded-treewidth Models through Guardedness
  - Joining Acyclicity and Guardedness
  - Algorithmic Aspects
- DB View: Query rewriting
- Mixing the Views
Acyclicity → Finite Universal Models

- **Goal:** find a simple syntactic way to guarantee that bottom-up rule applications lead to only finitely many invented values
  (such a set of rules is also called *fes* – *finite extension set*)

- **Idea:**
  - Trace the predicate positions that an invented value can “move” to
  - Use this to estimate if the invented value could (indirectly) trigger the rule that it was generated from
Example

\[
\begin{align*}
mother(x) & \rightarrow person(x) \\
person(x) \land mother(x) & \rightarrow \exists v. \text{parent}(v,x) \land person(v) \\
sibling(x,y) & \rightarrow \exists w. \text{parent}(x,w) \land \text{parent}(y,w) \land mother(w)
\end{align*}
\]
Weak Acyclicity

mother(x) → person(x)

person(x) ∧ mother(x) → ∃v.parent(v,x) ∧ person(v)

sibling(x,y) → ∃w.parent(x,w) ∧ parent(y,w) ∧ mother(w)
Joint Acyclicity

mother(x) → person(x)

person(x) ∧ mother(x) → ∃ v. parent(v,x) ∧ person(v)

sibling(x,y) → ∃ w. parent(x,w) ∧ parent(y,w) ∧ mother(w)
Joint Acyclicity

\[
\begin{align*}
mother(x) & \rightarrow person(x) \\
person(x) \land mother(x) & \rightarrow \exists v. parent(v,x) \land person(v) \\
sibling(x,y) & \rightarrow \exists w. parent(x,w) \land parent(y,w) \land mother(w)
\end{align*}
\]

\(\Omega_v: parent/1\)
Joint Acyclicity

mother(x) → person(x)

person(x) ∧ mother(x) → ∃v.parent(v,x) ∧ person(v)

sibling(x,y) → ∃w.parent(x,w) ∧ parent(y,w) ∧ mother(w)

Ω_v: parent/1, person/1
Joint Acyclicity

\[ \text{mother}(x) \rightarrow \text{person}(x) \]

\[ \text{person}(x) \land \text{mother}(x) \rightarrow \exists v. \text{parent}(v,x) \land \text{person}(v) \]

\[ \text{sibling}(x,y) \rightarrow \exists w. \text{parent}(x,w) \land \text{parent}(y,w) \land \text{mother}(w) \]

\[ \Omega_v : \text{parent}/1, \text{person}/1 \]

\[ \Omega_w : \text{parent}/2, \text{mother}/1, \text{person}/1 \]
Joint Acyclicity

mother(x) → person(x)

person(x) ∧ mother(x) → ∃ v. parent(v,x) ∧ person(v)

sibling(x,y) → ∃ w. parent(x,w) ∧ parent(y,w) ∧ mother(w)

Ω_v: parent/1, person/1

Ω_w: parent/2, mother/1, person/1

W  V
Eliminating Jointly Acyclic Existential Quantifiers

\[ \text{Sibling}(x,y) \rightarrow \exists w. \text{parent}(x,w) \land \text{parent}(y,w) \land \text{mother}(w) \]
Eliminating Jointly Acyclic Existential Quantifiers

\[
\text{sibling}(x,y) \rightarrow \exists w. \text{parent}(x,w) \land \text{parent}(y,w) \land \text{mother}(w)
\]

- **Skolemize:**

\[
\text{sibling}(x,y) \rightarrow \text{parent}(x,f(x,y)) \land \text{parent}(y,f(x,y)) \land \\
\text{mother}(f(x,y))
\]
Eliminating Jointly Acyclic Existential Quantifiers

\[ \text{sibling}(x,y) \rightarrow \exists w. \text{parent}(x,w) \land \text{parent}(y,w) \land \text{mother}(w) \]

- **Skolemize:**

  \[ \text{sibling}(x,y) \rightarrow \text{parent}(x,\text{f}(x,y)) \land \text{parent}(y,\text{f}(x,y)) \land \text{mother(}\text{f}(x,y)) \]

- But **function symbols** difficult to handle computationally
- Get rid of functions by **flattening:**

  \[ \text{sibling}(x,y) \rightarrow \text{parent}(x,\text{f}(x,y)) \land \text{parent}(y,\text{f}(x,y)) \land \text{mother(}\text{f}(x,y)) \]
Flattened Example

Arity of predicates increases everywhere:

\[ \text{mother}(x,x_1,x_2) \rightarrow \text{person}(x,x_1,x_2) \]

\[ \text{person}(x,x_1,x_2) \land \text{mother}(x,x_1,x_2) \rightarrow \exists v. \text{parent}(v,x,x_1,x_2) \land \text{person}(v,\Box,\Box) \]

\[ \text{sibling}(x,y) \rightarrow \text{parent}(x,f,x,y) \land \text{parent}(y,f,x,y) \land \text{mother}(f,x,y) \]
Can this Actually Work?

- Flattening can only represent terms of depth up to 1
- Acyclicity controls nesting of Skolem functions symbols
- Flattening based on topological order of variables in existential dependency graph
- Transformation can increase predicate arity exponentially

Today, a plethora of more acyclicity notions exist. Typically there is a trade-off between coverage and cost of checking.
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
- KR View: KB Rewriting into Nice Models
  - Finite models through Acyclicity
  - Bounded-treewidth Models through Guardedness
  - Joining Acyclicity and Guardedness
  - Algorithmic Aspects
- DB View: Query rewriting
- Mixing the Views
Essentially [Cali Gottlob Kifer KR’08]

\(\Sigma\) is \textit{bts} (bounded treewidth set) if for any database \(D\), the chase with \(\Sigma\) generates a structure with finite treewidth

i.e., for any database \(D\), there is an integer \(b\) such that \(\text{chase}(D,\Sigma)\) has treewidth of \(b\).

All \(\Sigma\) with finite chase are included in \textit{bts}
(bound given by the number of terms in the finite chase)
Decomposition tree:
1) each node (term) appears in a bag
2) each hyperedge (atom) has all its nodes in a bag
3) for each node $x$, the subgraph induced by the bags containing $x$ is connected

Width of a tree decomposition = max number of nodes in a bag (minus 1)
Treewidth of a graph = min width over all decomposition trees of this graph
Guardedness → Universal Models of Bounded Treewidth

**General Idea:** variables of a rule that might represent invented values must occur together in one body atom
The Family of Guarded Rules (Selection)

**Frontier:** variables shared by the body and the head

Guard only the *frontier*

The *frontier* has size 1

Guard only *affected* variables from the *frontier*

Guard only *affected* variables (i.e. possibly mapped to new existentials)

An atom in the body *guards* all the body variables

---

**Frontier1**

- \( r(x,y) \land r(y,z) \Rightarrow r(z,u) \)

**guarded**

- \( r(x,y) \land r(y,z) \land s(x,y,z) \Rightarrow r(y,u) \land r(z,u) \)

**weakly guarded**

- \( r(x,y) \land r(y,z) \Rightarrow r(z,u) \)

- \( r(x,y) \land r(y,z) \Rightarrow r(y,u) \land r(z,u) \)

**weakly frontier guarded**

- \( r(x,y) \land r(y,z) \Rightarrow r(y,u) \land r(z,u) \)

- \( r(x,y) \land r(y,z) \land s(x,y,z) \Rightarrow r(y,u) \land r(z,u) \)

---

[Cali+ KR' 08]

[Datalog]
Example

c(x) \land \text{ancestor}(x,y) \land \text{ancestor}(y,z) \rightarrow \text{ancestor}(x,z)

\text{parent}(x,y) \rightarrow \text{ancestor}(x,y)

c(x) \rightarrow \text{person}(x)

\text{person}(x) \rightarrow \exists w.\text{parent}(x,w) \land \text{person}(w)

\text{sibling}(x,y) \rightarrow \exists v.\text{parent}(x,v) \land \text{parent}(y,v) \land c(v)

\text{parent}(x,y) \land \text{sibling}(y,z) \rightarrow \text{uncle}(x,z)
Guardedness

c(x) ∧ ancestor(x,y) ∧ ancestor(y,z) → ancestor(x,z)

parent(x,y) → ancestor(x,y)

c(x) → person(x)

person(x) → ∃w.parent(x,w) ∧ person(w)

sibling(x,y) → ∃v.parent(x,v) ∧ parent(y,v) ∧ c(v)

parent(x,y) ∧ sibling(y,z) → uncle(x,z)
Frontier-Guardedness

c(x) \land \text{ancestor}(x,y) \land \text{ancestor}(y,z) \rightarrow \text{ancestor}(x,z)

\text{parent}(x,y) \rightarrow \text{ancestor}(x,y)

c(x) \rightarrow \text{person}(x)

\text{person}(x) \rightarrow \exists w. \text{parent}(x,w) \land \text{person}(w)

\text{sibling}(x,y) \rightarrow \exists v. \text{parent}(x,v) \land \text{parent}(y,v) \land c(v)

\text{parent}(x,y) \land \text{sibling}(y,z) \rightarrow \text{uncle}(x,z)
Weak Guardedness

c(x) \land \text{ancestor}(x,y) \land \text{ancestor}(y,z) \rightarrow \text{ancestor}(x,z)

\text{parent}(x,y) \rightarrow \text{ancestor}(x,y)

c(x) \rightarrow \text{person}(x)

\text{person}(x) \rightarrow \exists w.\text{parent}(x,w) \land \text{person}(w)

\text{sibling}(x,y) \rightarrow \exists v.\text{parent}(x,v) \land \text{parent}(y,v) \land c(v)

\text{parent}(x,y) \land \text{sibling}(y,z) \rightarrow \text{uncle}(x,z)
Weak Frontier-Guardedness

c(x) \land \text{ancestor}(x,y) \land \text{ancestor}(y,z) \rightarrow \text{ancestor}(x,z)

parent(x,y) \rightarrow \text{ancestor}(x,y)

c(x) \rightarrow \text{person}(x)

\text{person}(x) \rightarrow \exists w.\text{parent}(x,w) \land \text{person}(w)

\text{sibling}(x,y) \rightarrow \exists v.\text{parent}(x,v) \land \text{parent}(y,v) \land c(v)

parent(x,y) \land \text{sibling}(y,z) \rightarrow \text{uncle}(x,z)
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
- KR View: KB Rewriting into Nice Models
  - Finite models through Acyclicity
  - Bounded-treewidth Models through Guardedness
  - Joining Acyclicity and Guardedness
- Algorithmic Aspects
- DB View: Query rewriting
- Mixing the Views
Reconciling Acyclicity and Guardedness

- **Idea**: No need to guard invented values that don't occur in existential dependency cycles

**Approach for reasoning:**

- Eliminate all jointly acyclic existential variables by flattening
- Process remaining (guarded) existential variables as usual
Example

c(x) \land \text{ancestor}(x,y) \land \text{ancestor}(y,z) \rightarrow \text{ancestor}(x,z)

parent(x,y) \rightarrow \text{ancestor}(x,y)

c(x) \rightarrow \text{person}(x)

\text{person}(x) \rightarrow \exists w.\text{parent}(x,w) \land \text{person}(w)

\text{sibling}(x,y) \rightarrow \exists v.\text{parent}(x,v) \land \text{parent}(y,v) \land c(v)

\text{parent}(x,y) \land \text{sibling}(y,z) \rightarrow \text{uncle}(x,z)
Glut-Guardedness

c(x) ∧ ancestor(x,y) ∧ ancestor(y,z) → ancestor(x,z)

parent(x,y) → ancestor(x,y)

c(x) → person(x)

person(x) → ∃w.parent(x,w) ∧ person(w)

sibling(x,y) → ∃v.parent(x,v) ∧ parent(y,v) ∧ c(v)

parent(x,y) ∧ sibling(y,z) → uncle(x,z)
Glut-Frontier-Guardedness

\[ c(x) \land \text{ancestor}(x,y) \land \text{ancestor}(y,z) \rightarrow \text{ancestor}(x,z) \]

\[ \text{parent}(x,y) \rightarrow \text{ancestor}(x,y) \]

\[ c(x) \rightarrow \text{person}(x) \]

\[ \text{person}(x) \rightarrow \exists w. \text{parent}(x,w) \land \text{person}(w) \]

\[ \text{sibling}(x,y) \rightarrow \exists v. \text{parent}(x,v) \land \text{parent}(y,v) \land c(v) \]

\[ \text{parent}(x,y) \land \text{sibling}(y,z) \rightarrow \text{uncle}(x,z) \]
Expressivities and Complexities

- glut-guarded
- jointly acyclic
- jointly guarded
- weakly acyclic
- weakly guarded
- guarded

- glut-frontier-guarded
- jointly frontier-guarded
- weakly frontier-guarded
- frontier-guarded

Query complexity:
- ExpTime
- 2ExpTime
- 3ExpTime

Technische Universität Dresden
Computational Logic Group
Institute for Artificial Intelligence

22.09.2015
Sebastian Rudolph
Query Answering under Existential Rules
ICCL Summer School 2015
Main insight:

Acyclicity can be combined with any other decidability principle in a modular way and the result is more expressive than the sum of its parts.
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
- KR View: KB Rewriting into Nice Models
  - Finite models through Acyclicity
  - Bounded-treewidth Models through Guardedness
  - Joining Acyclicity and Guardedness
  - Algorithmic Aspects
- DB View: Query rewriting
- Mixing the Views
Worst-Case Optimal Reasoning with Guarded Existential Rules

- decidability of bounded-treewidth sets comes with a suboptimal algorithm
- desirable: worst-case optimal algorithm(s) for known versions of guardedness
- to this end:
  - introduce new class: greedy bounded-treewidth sets of rules (gbts) subsuming all presented versions of guarded rules
  - provide algorithm for this class
Greedy Bounded-Treewidth Sets of Rules (gbts)

Allow for a greedy construction of a decomposition tree in the course of the chase

\[ T_0 \cup \text{var}(s(H)) \]

Gbts condition:
Frontier variables that are not being mapped to terms from D are \textit{jointly} mapped to variables occurring in atoms added by a single rule application.

\[ T_0 = \text{terms}(D) + \{\text{constants}\} \]

All bags contain \( T_0 \)
Greedy \textit{bts}

\begin{align*}
R_1 & : \ p(x,y) \rightarrow q(y,z) \\
R_2 & : \ p(x,y) \land q(y,z) \rightarrow r(x,y,t) \land p(y,t)
\end{align*}

D = p(a,b)

Greedy construction of a \textit{decomposition tree} of the chase

with bounded width
Worst-Case Optimal Reasoning with Guarded Existential Rules

- for gbts, chase can be “lifted” to the decomposition tree
- introduce *patterns* describing the (finitely many) ways one bag of the tree decomposition may look like (similar to “types” used in logics)
- apply a consequence-based technique to construct the initial part of the greedy tree decomposition, terminate as soon as type-repetition is detected (similar to *blocking* in DL tableaux)
Worst-Case Optimal Reasoning with Guarded Existential Rules

“full blocked tree” represents the full chase and can be used for CQ answering
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
- KR View: KB Rewriting into Nice Models
  - Finite models through Acyclicity
  - Bounded-treewidth Models through Guardedness
  - Joining Acyclicity and Guardedness
  - Algorithmic Aspects
- DB View: Query rewriting
- Mixing the Views
rewrite ontology-mediated query into a representation that is easier to execute against the data

- frequent choice: Unions of Conjunctive Queries
- other options: (fragments of) Datalog or disjunctive Datalog
Backward Chaining Scheme

Basic step:

Query

Rule

Body

Head

Unification

Query rewriting

New query
Finite Unification Sets of Rules

- Iterated backward chaining creates a union of CQs (UCQ).
- Terminates if after one rewriting step (w.r.t. all rules) the obtained UCQ is subsumed by the previous one.
- If, given a $\Sigma$, termination of backward chaining is guaranteed for all possible Q, we call $\Sigma$ finite unification set (fus) or first-order rewritable.
- This property is not decidable, but again, syntactic criteria have been identified: DL-Lite, sticky rules, sticky-join rules,…
Outline

- Classical Query Answering
  - Some Remarks about Query Languages
- Ontological Query Answering: Two Views
- KR View: KB Rewriting into Nice Models
  - Finite models through Acyclicity
  - Bounded-treewidth Models through Guardedness
  - Joining Acyclicity and Guardedness
  - Algorithmic Aspects
- DB View: Query rewriting
- Mixing the Views
Views on Ontological Query Answering

The Mixed View:

\[ D \models_\Sigma Q \]
The Mixed View:

\[
D \models_{\Sigma_1 + \Sigma_2} Q
\]
Views on Ontological Query Answering

The Mixed View:

\[(D, \Sigma_1) \models (\Sigma_2, Q)\]
The Mixed View:

(Knowledge Base, \(\Sigma_1\)) \models (\Sigma_2, Q)

- Knowledge Base
- \(\Sigma_1\)
- Data \(D\)
- Ontology-mediated Query
- \(\Sigma_2\)
- Query \(Q\)
A tool: the Graph of Rule Dependencies

[R2 depends on R1 if applying R1 may trigger a new application of R2]

i.e., there exists a database D s.t. R1 is applicable to D but R2 is not and there is an application of R1 to D leading to D’ s.t. R2 is applicable to D’

Effective computation of dependencies with a piece-unification test:

R2 depends on R1 iff there is a piece-unifier of body(R2) with head(R1)
Combining Decidable Classes with the Graph of Rule Dependencies

Rules

\[ R_1 \rightarrow R_2 : R_1 \text{ «may trigger» } R_2 \text{ (} R_2 \text{ depends on } R_1 \text{)} \]
Combining Decidable Classes with the Graph of Rule Dependencies

If GRD(Σ) is **without cycle** then Σ is both fes (thus bts) and fus

\[
\text{fes} = \text{finite chase} \\
\text{fus} = \text{finite conjunctive query rewriting} \\
\text{bts} = (\text{possibly infinite}) \text{ bounded-treewidth chase}
\]
If all strongly connected components of GRD(Σ) are fes then Σ is fes [Baget 2004]

The same holds for fus (but not for bts)
Combining Decidable Classes with the Graph of Rule Dependencies

Let $\Sigma_1, \Sigma_2$ be a partition of $\Sigma$ s.t. no rule of $\Sigma_1$ depends on a rule of $\Sigma_2$

- If $\Sigma_1$ is $fes$ and $\Sigma_2$ is $bts$, then $\Sigma$ is $bts$
- If $\Sigma_1$ is $bts$ and $\Sigma_2$ is $fus$, then $\Sigma$ is decidable
Conclusion

- currently there are two antagonistic computational approaches to ontological query answering:
  - forward chaining starting from data
  - backward chaining starting from query
- both approaches come with (abstract) criteria and concrete syntactic conditions that ensure decidability
- approaches can be “loosely coupled” if ontology allows for adequate separation
- desirable: a common criterion subsuming both
  → future work
Special Thanks & Credits

Presented contributions joint work with a lot of colleagues (and special thanks to Markus, Marie-Laure and Michaël for letting me use their slides)