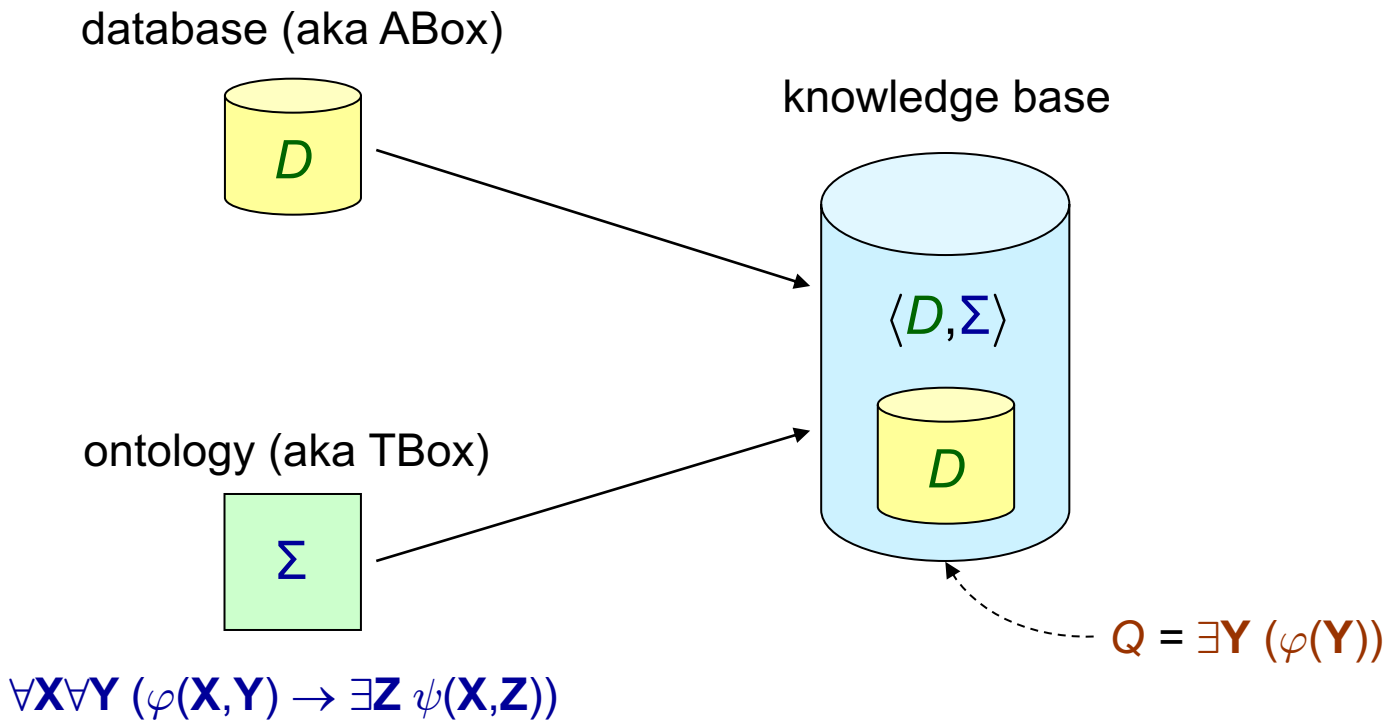
Sebastian Rudolph
International Center for Computational Logic
TU Dresden

Existential Rules – Lecture 9

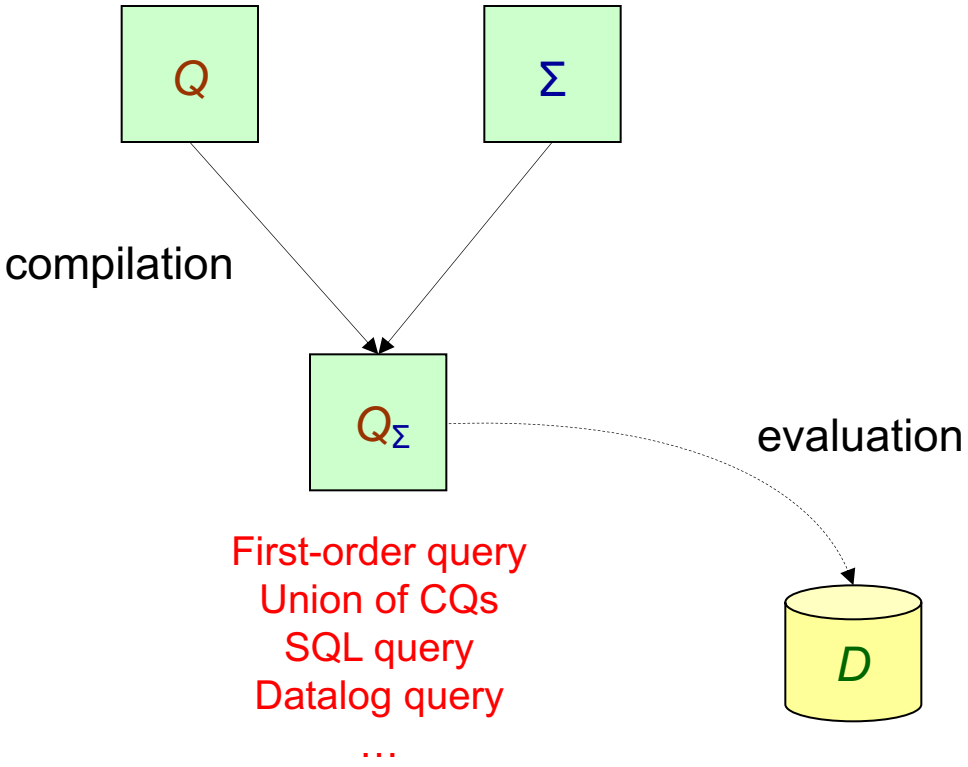
Adapted from slides by Andreas Pieris and Michaël Thomazo
Summer Term 2023

BCQ-Answering: Our Main Decision Problem



decide whether $D \wedge \Sigma \models Q$

Query Rewriting



$\forall D : D \wedge \Sigma \models Q \quad \Leftrightarrow \quad D \models Q_\Sigma$

evaluated and optimized by

exploiting existing technology

Query Rewriting: Formal Definition

Consider a class of existential rules $\mathcal{L}$, and a query language $\mathcal{Q}$.

BCQ-Answering under $\mathcal{L}$ is $\mathcal{Q}$-rewritable if, for every $\Sigma \in \mathcal{L}$ and BCQ Q,

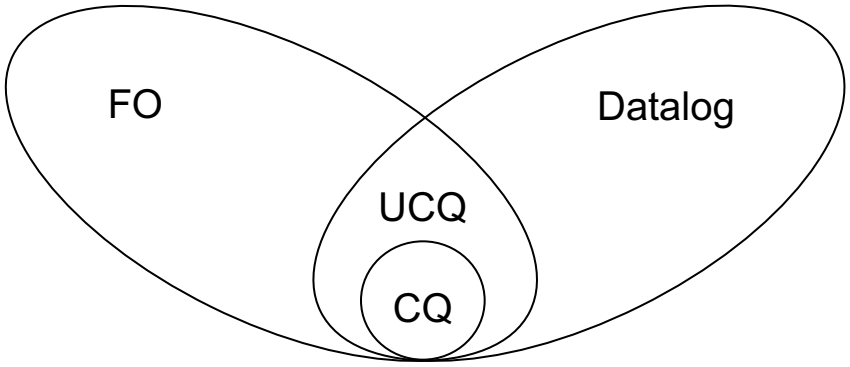
we can construct a query $Q_\Sigma \in \mathcal{Q}$ such that,

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

NOTE: The construction of Q_Σ is database-independent – the pure approach to query rewriting

Target Query Language

we target the weakest query language



	CQ	UCQ	FO	Datalog
FULL	✗	✗	✗	✓
ACYCLIC	✗	✓	✓	✓
LINEAR	✗	✓	✓	✓

UCQ-Rewritings

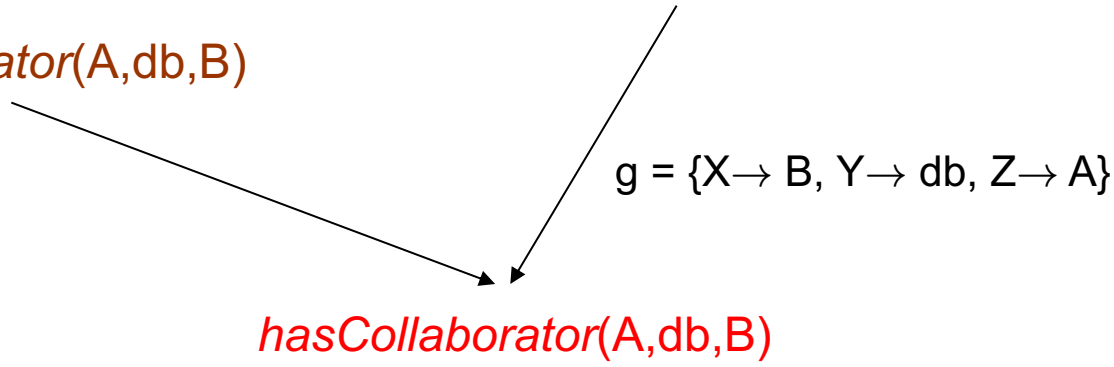
- The standard algorithm for computing UCQ-rewritings performs an exhaustive application of the following two steps:
    1. Rewriting
    2. Minimization

- The standard algorithm is designed for normalized existential rules, where only one atom appears in the head
  (any ruleset can be normalized while preserving the query answers; normalization also does not destroy acyclicity/linearity)


Rewriting Step

$\Sigma = \{\forall X \forall Y\ (\mathit{project}(X) \wedge \mathit{inArea}(X,Y) \rightarrow \exists Z\ \mathit{hasCollaborator}(Z,Y,X))\}$

$Q = \exists A \exists B\ \mathit{hasCollaborator}(A,db,B)$



Thus, we can simulate a “backward chase step” by a resolution step

$Q_\Sigma = \exists A \exists B\ \mathit{hasCollaborator}(A,db,B)$

$\vee$

$\exists B\ (\mathit{project}(B) \wedge \mathit{inArea}(B,db))$

Applicability Condition

Consider a BCQ Q, an atom α in Q, and a (normalized) rule σ.

We say that σ is applicable to α if the following conditions hold:

1. head(σ) and α unify via h : terms(head(σ)) $\rightarrow$ terms(α)

2. For every variable X in head(σ), if h(X) is a constant, then X is a $\forall$-variable

3. For every variable X in head(σ), if h(X) = h(Y), where Y is a shared variable of α, then X is a $\forall$-variable

4. If X is an $\exists$-variable of head(σ), and Y is a variable in head(σ) such that X $\neq$ Y, then h(X) $\neq$ h(Y)

...but, although this is crucial for soundness, it may destroy completeness

The Rewriting Algorithm

Q_Σ := Q;
repeat
  Q_{aux} := Q_Σ;
  foreach disjunct q of Q_{aux} do
  //Rewriting Step
    foreach atom α in q do
      foreach rule σ in Σ do
        if σ is applicable to α then
          q_{rew} := *rewrite*(q,α,σ); // resolve α using σ
          if q_{rew} does not appear in Q_Σ (modulo variable renaming) then
            Q_Σ := $Q_\Sigma \vee q_{rew}$;
  //Minimization Step
    foreach pair of atoms α,β in q that unify do
      q_{min} := *minimize*(q,α,β); // apply most general unifier of α and β on q
      if q_{min} does not appear in Q_Σ (modulo variable renaming) then
        Q_Σ := $Q_\Sigma \vee q_{min}$;
until Q_{aux} = Q_Σ;
return Q_Σ;

Termination

Theorem: The rewriting algorithm terminates under ACYCLIC and LINEAR

Proof (ACYCLIC):

- 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 ACYCLIC and LINEAR

Proof (LINEAR):

- 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 $|Q| \cdot$ 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

Complexity of BCQ-Answering

	Data Complexity	
FULL	PTIME-c	Naïve algorithm
		Reduction from Monotone Circuit Value problem
ACYCLIC	in LOGSPACE	**UCQ-rewriting**
LINEAR		

	Combined Complexity	
FULL	EXPTIME-c	Naïve algorithm
		Simulation of a deterministic exponential time TM
ACYCLIC	NEXPTIME-c	Small witness property
		Reduction from Tiling problem
LINEAR	PSPACE-c	Level-by-level non-deterministic algorithm
		Simulation of a deterministic polynomial space TM

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!!!

$$\Sigma = \{\forall X\,(R_k(X) \rightarrow P_k(X))\}_{k \in \{1,\ldots,n\}} \qquad Q = \exists X\,(P_1(X) \wedge \ldots \wedge P_n(X))$$

$$\exists X\,(P_1(X) \wedge \ldots \wedge P_n(X))$$

$$P_1(X) \vee R_1(X) \qquad P_n(X) \vee R_n(X)$$

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

$$\forall D : D \wedge \Sigma \models Q \quad \Leftrightarrow \quad D \models Q_\Sigma$$

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

Last Words: The Bigger Picture