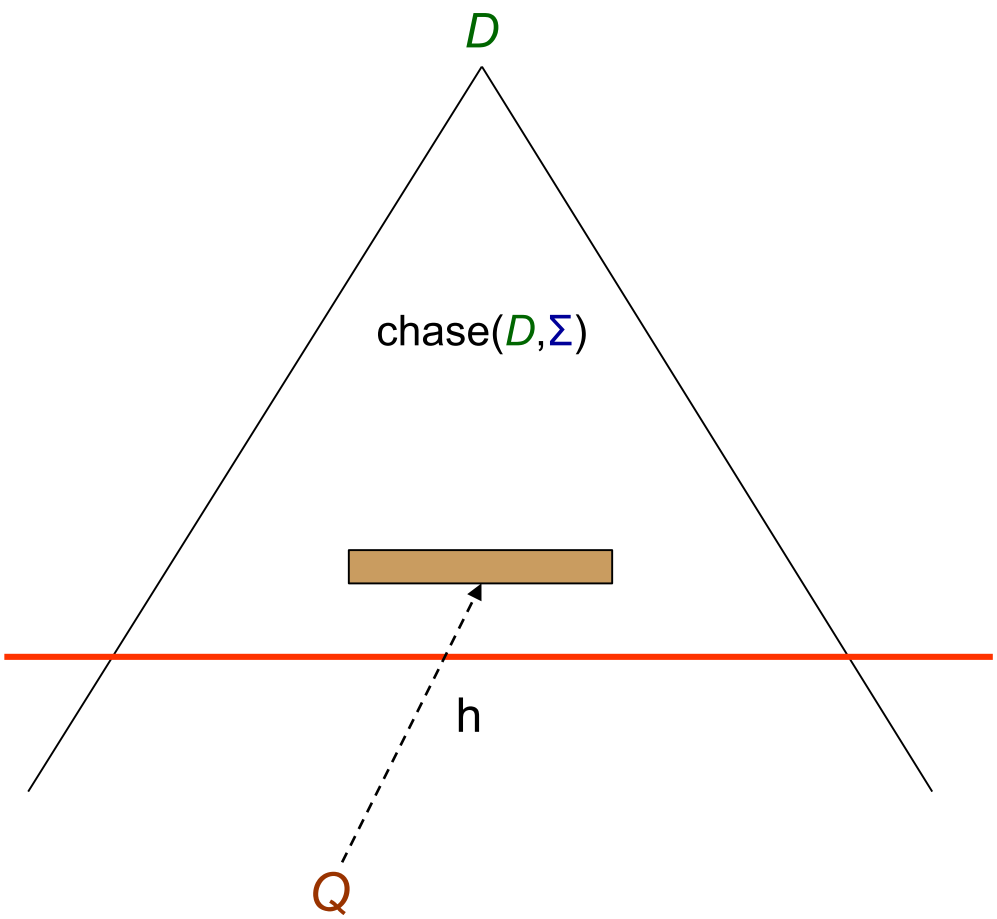
Query Rewriting in OBDA

Forward Chaining Techniques



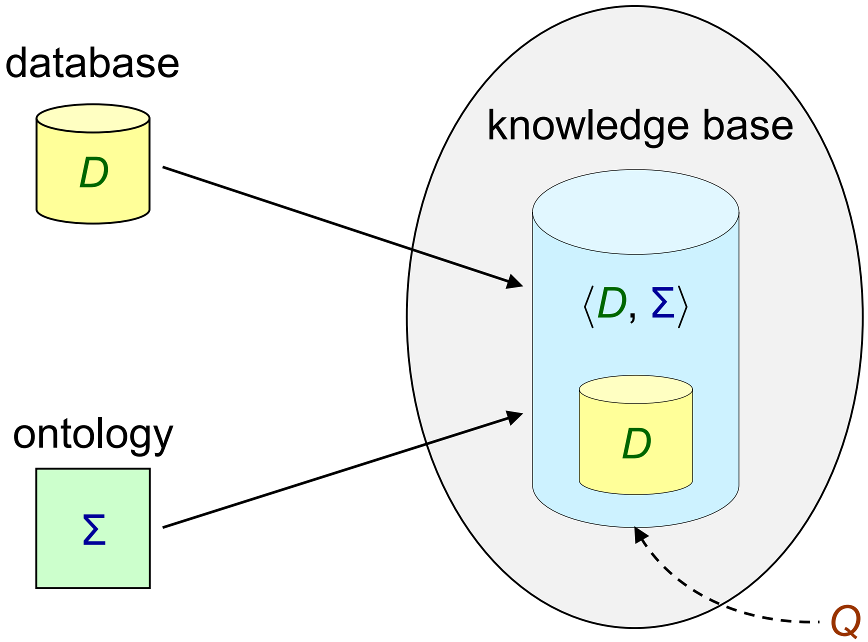
Useful techniques for establishing optimal upper bounds

…but not practical - we need to store instances of very large size

How we achieve true scalability in OBQA?

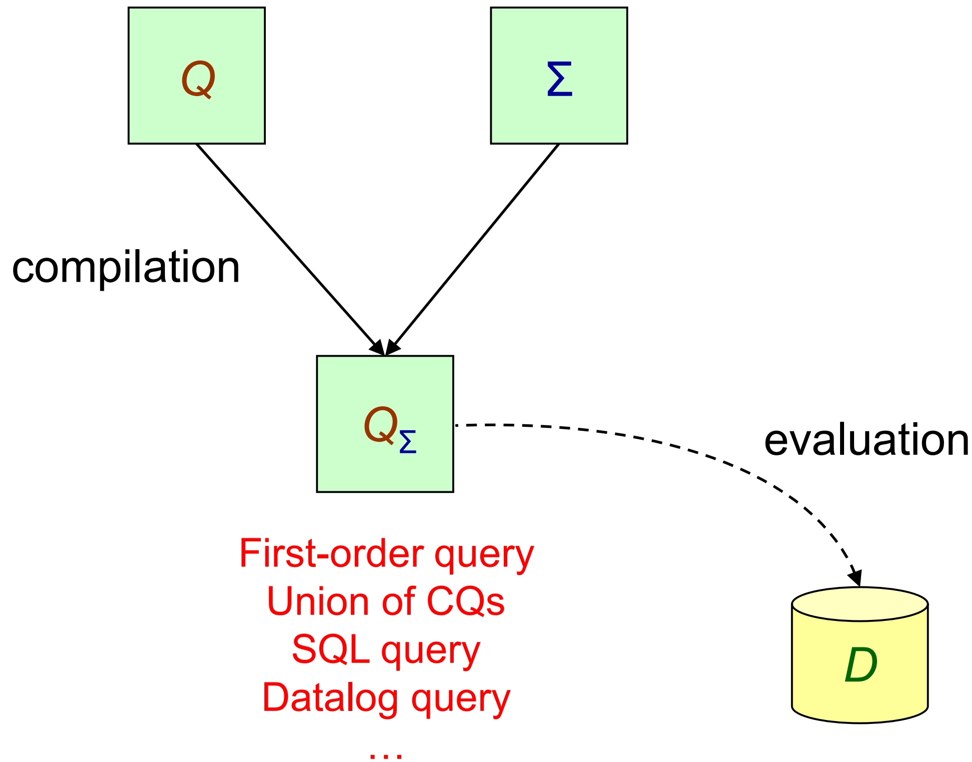
Scalability in OBQA

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



$$\forall D \;:\; D \wedge \Sigma \models Q \quad \Leftrightarrow \quad \boxed{D \models Q_\Sigma}$$

evaluated and optimized by

exploiting existing technology

Query Rewriting: Formal Definition

Consider a class of existential rules **L**, and a query language **Q**.

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, $D \wedge \Sigma \vDash Q$ iff $D \vDash Q_\Sigma$

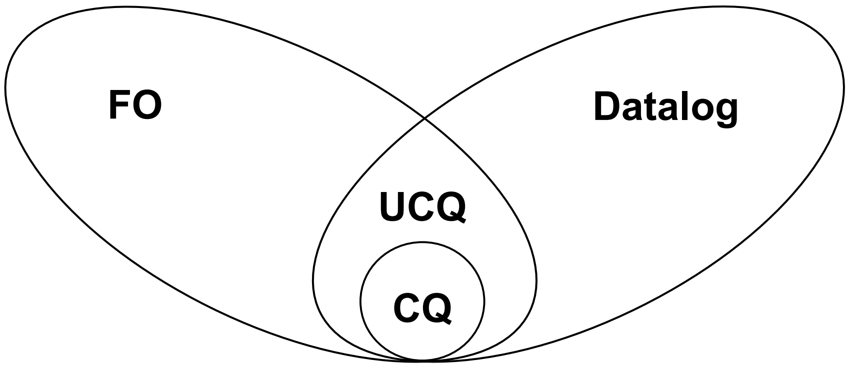
NOTE: The construction of Q_Σ is database-independent - the pure approach to query rewriting

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

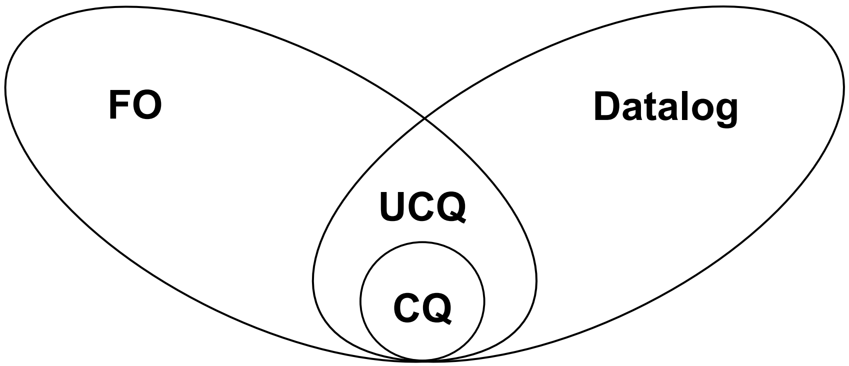
we target the weakest query language



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

Target Query Language

we target the weakest query language



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

Target Query Language

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

Q :- $S(x), U(x,y), T(y)$

$Q_\Sigma = \{Q$:- $S(x), U(x,y), T(y),$

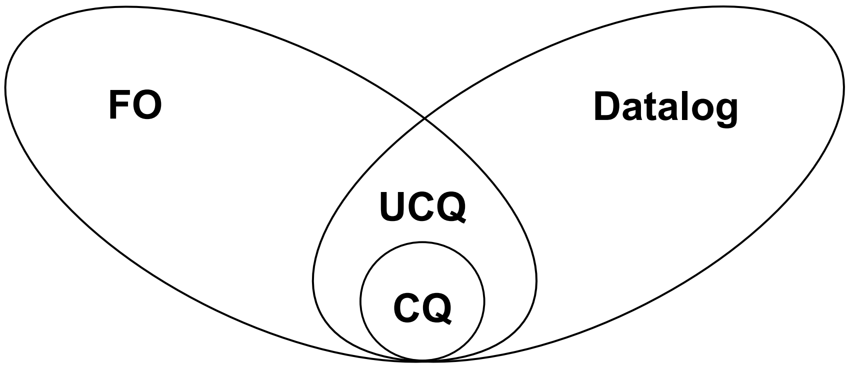
Q_1 :- $S(x), U(x,y), P(y),$

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



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

Target Query Language

$\Sigma = \{\forall x \forall y\ (R(x,y) \wedge P(y) \rightarrow P(x))\}$

Q :- $P(c)$

$$Q_\Sigma = \{Q :- P(c),$$

$$Q_1 :- R(c,y_1), P(y_1),$$

$$Q_2 :- R(c,y_1), R(y_1,y_2), P(y_2),$$

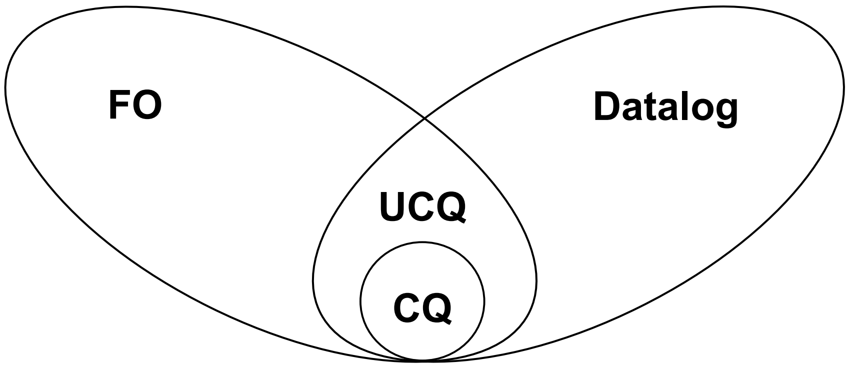
$$Q_3 :- R(c,y_1), R(y_1,y_2), R(y_2,y_3), P(y_3),$$

$$\ldots\}$$

- 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



	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

Normalization Procedure

$$\forall \mathbf{x} \forall \mathbf{y} \, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z} \, (P_1(\mathbf{x},\mathbf{z}) \wedge \ldots \wedge P_n(\mathbf{x},\mathbf{z})))$$

$$\downarrow$$

$$\forall \mathbf{x} \forall \mathbf{y} \, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z} \, \text{Auxiliary}(\mathbf{x},\mathbf{z}))$$

$$\forall \mathbf{x} \forall \mathbf{z} \, (\text{Auxiliary}(\mathbf{x},\mathbf{z}) \rightarrow P_1(\mathbf{x},\mathbf{z}))$$

$$\forall \mathbf{x} \forall \mathbf{z} \, (\text{Auxiliary}(\mathbf{x},\mathbf{z}) \rightarrow P_2(\mathbf{x},\mathbf{z}))$$

$$\ldots$$

$$\forall \mathbf{x} \forall \mathbf{z} \, (\text{Auxiliary}(\mathbf{x},\mathbf{z}) \rightarrow P_n(\mathbf{x},\mathbf{z}))$$

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

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

Q :- hasCollaborator(u,db,v)

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

hasCollaborator(u,db,v)

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

$Q_\Sigma = \{Q$:- hasCollaborator(u,db,v),

Q_1 :- project(v), inArea(v,db)$\}$

Unsound Rewritings

$\Sigma = \{\forall x \forall y\ (project(x) \wedge inArea(x,y) \rightarrow \exists z\ hasCollaborator(z,y,x))\}$

Q :- hasCollaborator(c,db,v)

$g = \{x \rightarrow v,\ y \rightarrow db,\ z \rightarrow c\}$

hasCollaborator(c,db,v)

After applying the rewriting step we obtain the following UCQ

$Q_\Sigma = \{Q$:- hasCollaborator(c,db,v),

Q_1 :- project(v), inArea(v,db)$\}$

Unsound Rewritings

Σ = $\{\forall x \forall y\ (\text{project}(x) \land \text{inArea}(x,y) \rightarrow \exists z\ \text{hasCollaborator}(z,y,x))\}$

Q :- hasCollaborator(c,db,v)

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

- Consider the database D = {project(a), inArea(a,db)}
- Clearly, $D \vDash Q_\Sigma$
- However, $D \land \Sigma$ does not entail Q since there is no way to obtain an atom of the form hasCollaborator(c,db,_) during the chase

Unsound Rewritings

$\Sigma = \{\forall x \forall y\ (project(x) \wedge inArea(x,y) \rightarrow \exists z\ hasCollaborator(z,y,x))\}$

Q :- hasCollaborator(c,db,v)

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

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

Σ = $\{\forall x \forall y\ (project(x) \wedge inArea(x,y) \rightarrow \exists z\ hasCollaborator(z,y,x))\}$

Q :- hasCollaborator(v,db,v)

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

hasCollaborator(v,db,v)

After applying the rewriting step we obtain the following UCQ

Q_Σ = {Q :- hasCollaborator(v,db,v),

Q_1 :- project(v), inArea(v,db)}

Unsound Rewritings

$\Sigma = \{\forall x \forall y\ (\text{project}(x) \wedge \text{inArea}(x,y) \rightarrow \exists z\ \text{hasCollaborator}(z,y,x))\}$

Q :- hasCollaborator(v,db,v)

$Q_\Sigma = \{Q$:- hasCollaborator(v,db,v),

Q_1 :- project(v), inArea(v,db)$\}$

- Consider the database $D = \{$project(a), inArea(a,db)$\}$
- Clearly, $D \vDash Q_\Sigma$
- However, $D \wedge \Sigma$ does not entail Q since there is no way to obtain an atom of the form hasCollaborator(t,db,t) during the chase

Unsound Rewritings

Σ = $\{\forall x \forall y\ (project(x) \wedge inArea(x,y) \rightarrow \exists z\ hasCollaborator(z,y,x))\}$

Q :- hasCollaborator(v,db,v)

Q_Σ = $\{Q$:- hasCollaborator(v,db,v),

Q_1 :- project(v), inArea(v,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 ∃-variable

Applicability Condition

Consider a (Boolean) CQ 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

2. For every variable x in head(σ):
   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 α, then x is a $\forall$-variable

3. 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 is crucial for soundness, may destroy completeness

Incomplete Rewritings

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

$\qquad \forall x \forall y \forall z\ (\text{hasCollaborator}(x,y,z) \rightarrow \text{collaborator}(x))\}$

Q :- hasCollaborator(u,v,w), collaborator(u))

Q_Σ = {Q :- hasCollaborator(u,v,w), collaborator(u),

Q_1 :- hasCollaborator(u,v,w), hasCollaborator(u,v',w')

- Consider the database D = {project(a), inArea(a,db)}
- Clearly, chase(D,Σ) = $D \cup$ {hasCollaborator(z,db,a), collaborator(z)} $\vDash Q$
- However, D does not entail Q_Σ

Incomplete Rewritings

Σ = $\{\forall x \forall y\ (project(x) \wedge inArea(x,y) \rightarrow \exists z\ hasCollaborator(z,y,x)),$

$\forall x \forall y \forall z\ (hasCollaborator(x,y,z) \rightarrow collaborator(x))\}$

Q :- hasCollaborator(u,v,w), collaborator(u))

Q_Σ = {Q :- hasCollaborator(u,v,w), collaborator(u),

Q_1 :- hasCollaborator(u,v,w), hasCollaborator(u,v',w')

Q_2 :- project(u), inArea(u,v)

...but, we cannot obtain the last query due to the applicablity condition

Incomplete Rewritings

Σ = $\{\forall x \forall y\ (\text{project}(x) \wedge \text{inArea}(x,y) \rightarrow \exists z\ \text{hasCollaborator}(z,y,x)),$

$\forall x \forall y \forall z\ (\text{hasCollaborator}(x,y,z) \rightarrow \text{collaborator}(x))\}$

Q :- hasCollaborator(u,v,w), collaborator(u))

Q_Σ = $\{Q$:- hasCollaborator(u,v,w), collaborator(u),

Q_1 :- hasCollaborator(u,v,w), hasCollaborator(u,v',w')

Q_2 :- hasCollaborator(u,v,w) - by minimization

Q_3 :- project(w), inArea(w,v) - by rewriting

D = {project(a), 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_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);   //we resolve α using σ
                    if q_rew does not appear in Q_Σ (modulo variable renaming) then
                        Q_Σ := Q_Σ ∪ {q_rew};
    //Minimization Step
        foreach pair of atoms α,β in q that unify do
            q_min := minimize(q,α,β);   //we apply the MGU of α and β on q
            if q_min does not appear in Q_Σ (modulo variable renaming) then
                    Q_Σ := Q_Σ ∪ {q_min};
until Q_aux = Q_Σ;
return Q_Σ;
```

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

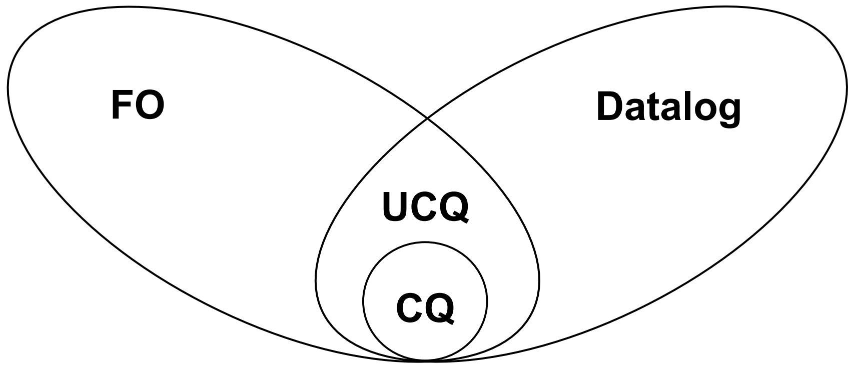
Theorem: The rewriting algorithm terminates under **LINEAR**

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 ($|Q| \cdot$ 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



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

Back to Complexity

	Data Complexity	
FULL	PTIME-c	Naïve algorithm
		Reduction from Monotone Circuit Value problem
ACYCLIC	in LOGSPACE	**Via 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 a 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 rewritng algorithm produces rewritings of exponential size
- Can we do better? **NO!!!**

$$\Sigma \;=\; \{\forall x\, (R_k(x) \rightarrow P_k(x))\}_{k \in \{1,\ldots,n\}} \qquad\qquad Q \;:\!-\; P_1(x), \ldots, P_n(x)$$

$$Q :\!- P_1(X), \ldots, P_n(X)$$

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

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 rewritng 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

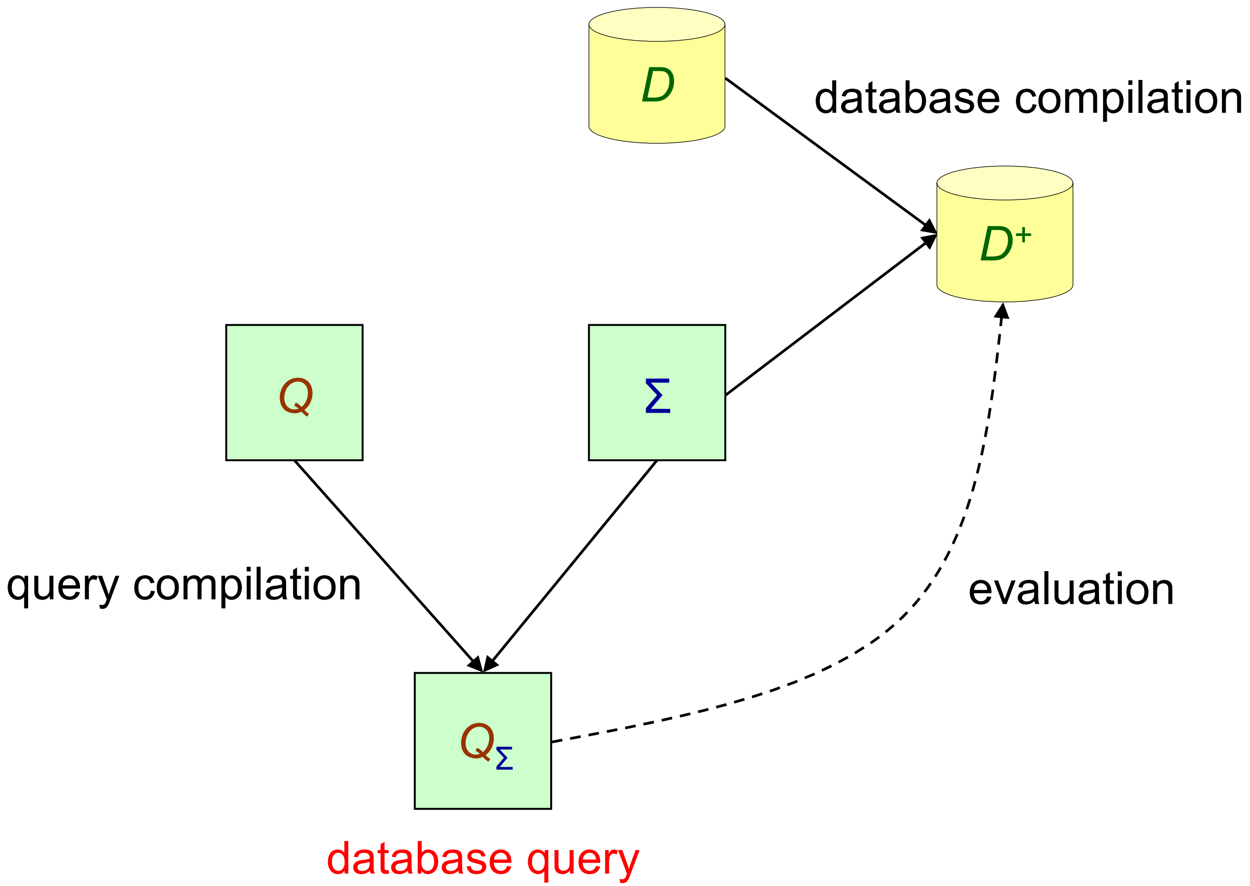
$\forall D : D \wedge \Sigma \models Q \quad \Leftrightarrow \quad D \models Q_\Sigma$

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

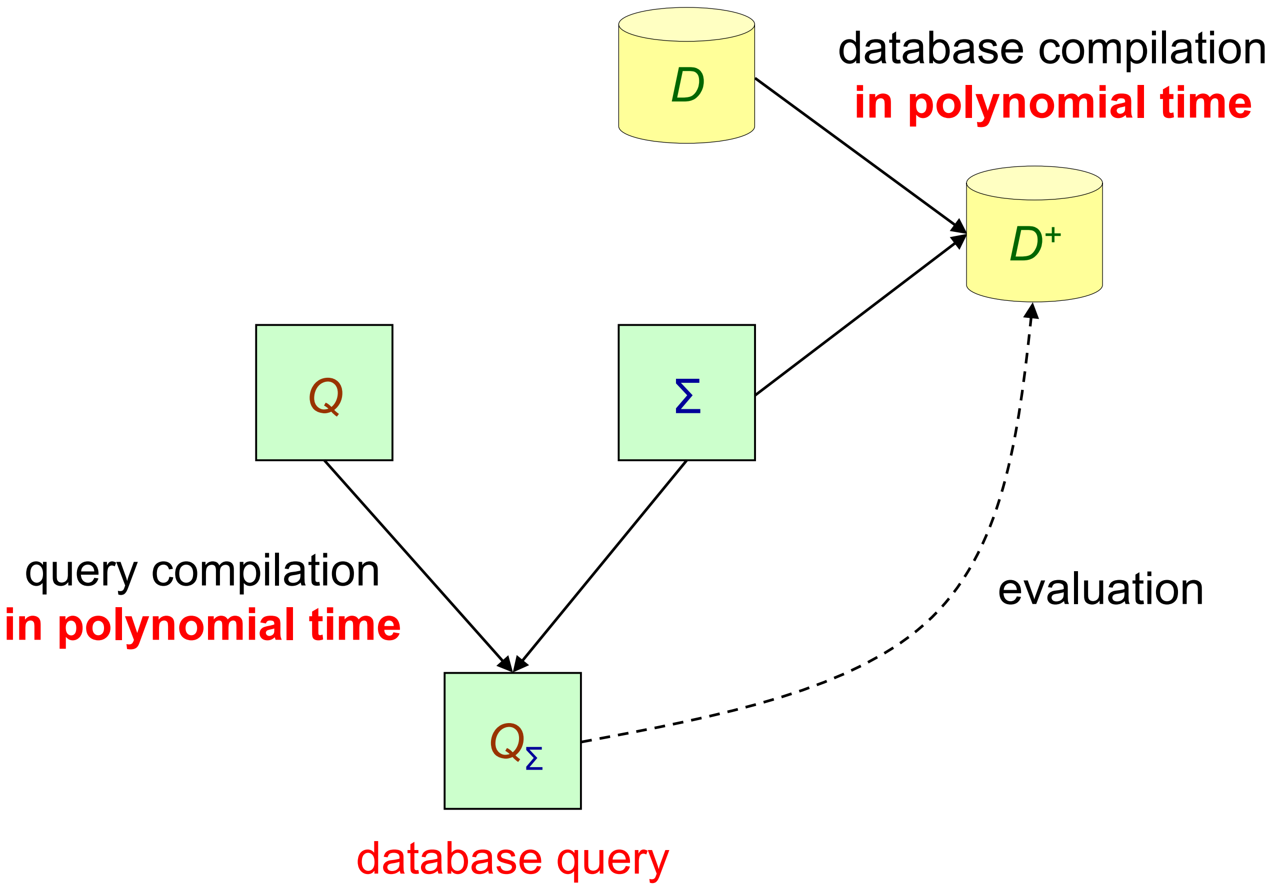
$\Rightarrow$ **the combined approach to query rewriting**

Combined Rewritability



$\forall D \; : \; D \wedge \Sigma \models Q \quad \Leftrightarrow \quad D^{+} \models Q_{\Sigma}$

Polynomial Combined Rewritability



$$\forall D : D \wedge \Sigma \models Q \quad \Leftrightarrow \quad D^{+} \models Q_{\Sigma}$$