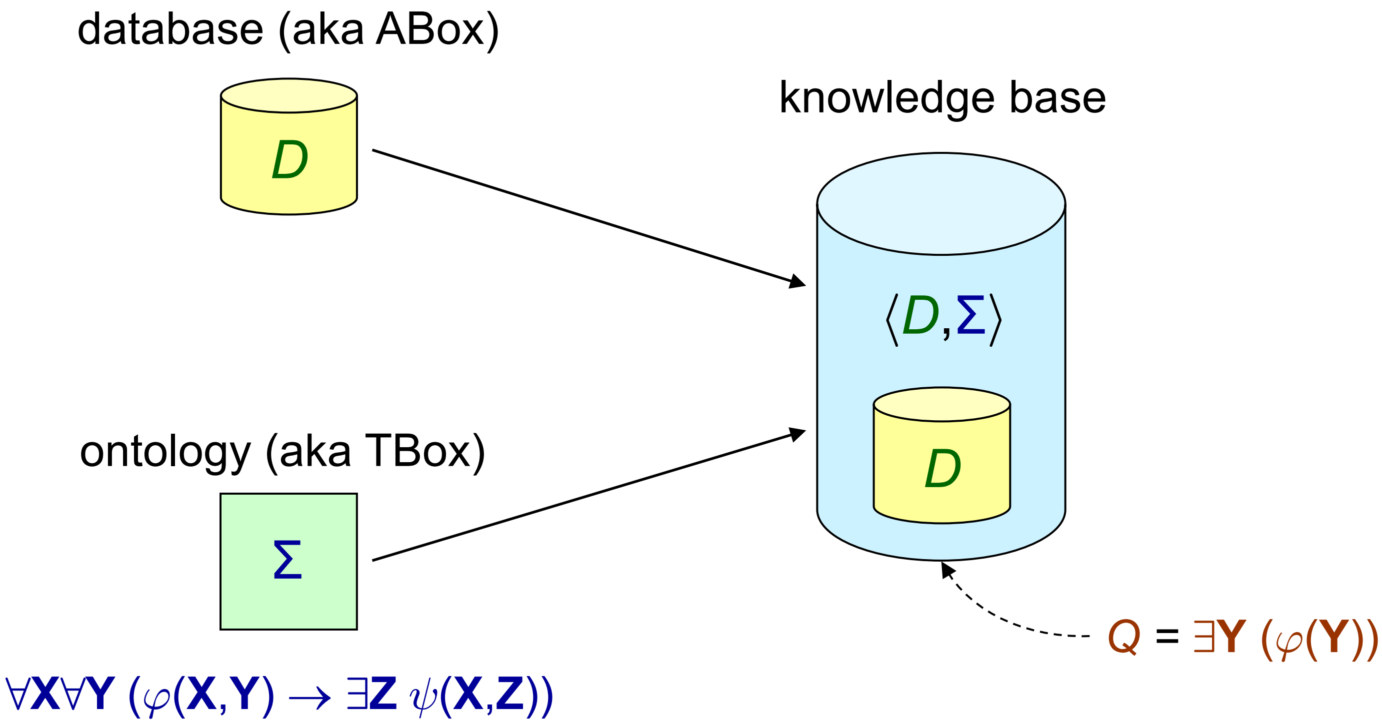
Sebastian Rudolph
International Center for Computational Logic
TU Dresden

Existential Rules – Lecture 8

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

BCQ-Answering: Our Main Decision Problem



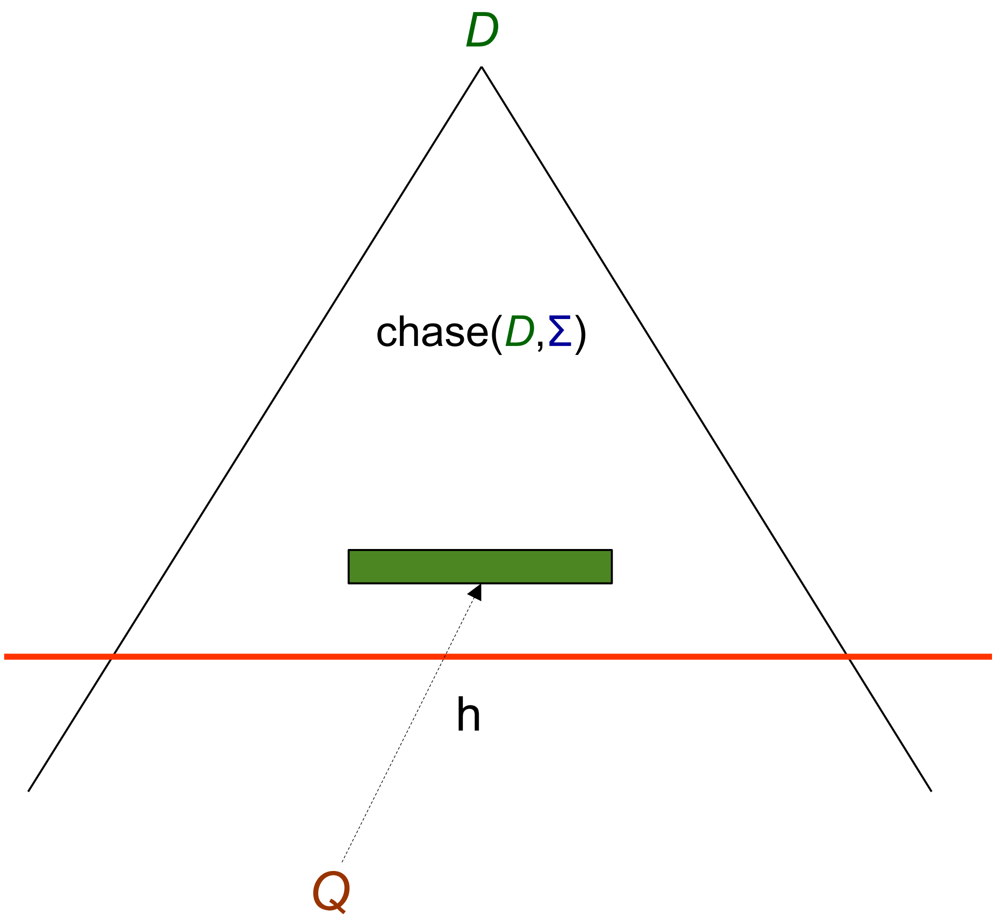
decide whether $D \wedge \Sigma \models Q$

Sum Up

	Data Complexity	
FULL	PTIME-c	Naïve algorithm
		Reduction from Monotone Circuit Value problem
ACYCLIC	in LOGSPACE	**Second part of our course**
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

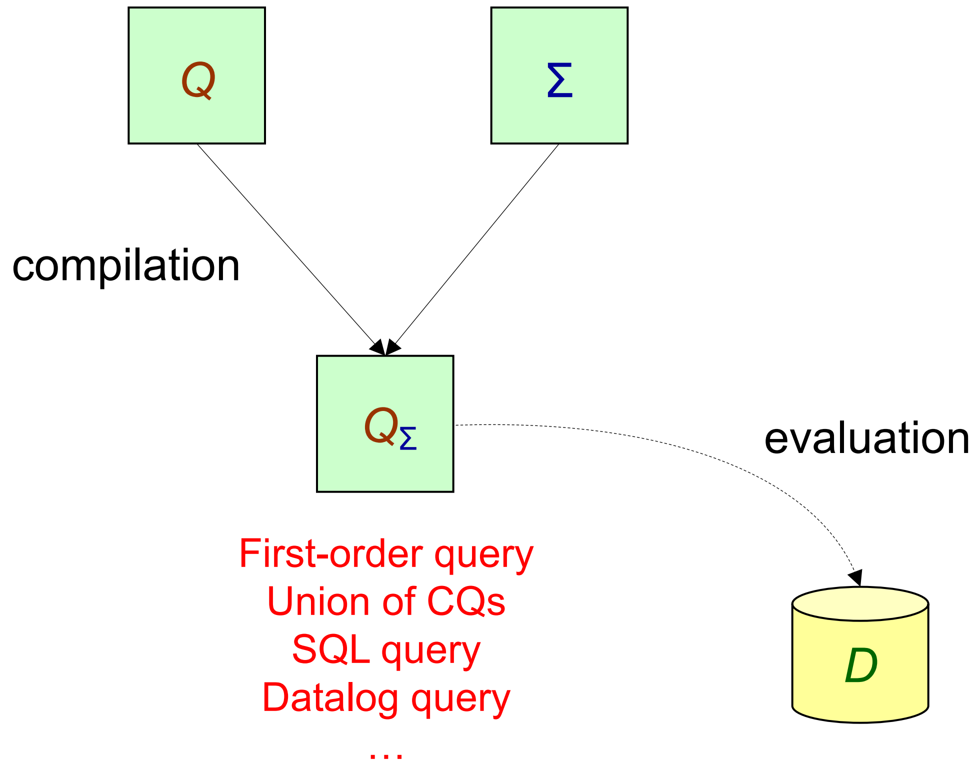
Forward Chaining Techniques



Useful techniques for establishing optimal upper bounds

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

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,

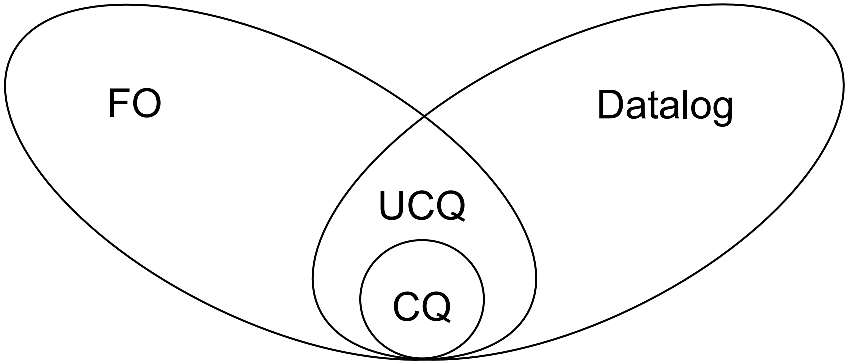
we can construct a query $Q_\Sigma \in \mathcal{Q}$ such that,

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

Normalization Procedure

$$\forall X \forall Y\ (\varphi(X,Y) \rightarrow \exists Z\ (P_1(X,Z) \wedge \ldots \wedge P_n(X,Z)))$$

$$\downarrow$$

$$\forall X \forall Y\ (\varphi(X,Y) \rightarrow \exists Z\ \mathit{Auxiliary}(X,Z))$$

$$\forall X \forall Z\ (\mathit{Auxiliary}(X,Z) \rightarrow P_1(X,Z))$$

$$\forall X \forall Z\ (\mathit{Auxiliary}(X,Z) \rightarrow P_2(X,Z))$$

$$\ldots$$

$$\forall X \forall Z\ (\mathit{Auxiliary}(X,Z) \rightarrow P_n(X,Z))$$

NOTE 1: Acyclicity and linearity are preserved

NOTE 2: We obtain an equivalent set w.r.t. query answering (not logically equivalent)

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

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

Q = $\exists A \exists B$ *hasCollaborator*(A,db,B)

g = {X$\rightarrow$ B, Y$\rightarrow$ db, Z$\rightarrow$ A}

hasCollaborator(A,db,B)

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

Q_{Σ} = $\exists A \exists B$ *hasCollaborator*(A,db,B)

$\vee$

$\exists B$ (*project*(B) $\wedge$ *inArea*(B,db))

Unsound Rewritings

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

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

$g = \{X \rightarrow B,\ Y \rightarrow db,\ Z \rightarrow c\}$

$\mathit{hasCollaborator}(c,db,B)$

After applying the rewriting step we obtain the following UCQ

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

$$\vee$$

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

Unsound Rewritings

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

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

$$\boxed{\begin{array}{l} Q_\Sigma = \exists B\, \mathit{hasCollaborator}(c,db,B) \\ \qquad\qquad\qquad \vee \\ \qquad \exists B\, (\mathit{project}(B) \wedge \mathit{inArea}(B,db)) \end{array}}$$

- Consider the database $D = \{\mathit{project}(a), \mathit{inArea}(a,db)\}$
- 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*(c,db,_) during the chase

Unsound Rewritings

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

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

$$\boxed{\begin{array}{l} Q_{\Sigma} = \exists B\ \mathit{hasCollaborator}(c,db,B) \\ \qquad\qquad \vee \\ \qquad \exists B\ (\mathit{project}(B) \wedge \mathit{inArea}(B,db)) \end{array}}$$

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

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

$g = \{X \rightarrow B,\ Y \rightarrow db,\ Z \rightarrow B\}$

$\mathit{hasCollaborator}(B,db,B)$

After applying the rewriting step we obtain the following UCQ

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

$$\vee$$

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

Unsound Rewritings

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

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

$$\boxed{\begin{array}{c} Q_\Sigma = \exists B\ \mathit{hasCollaborator}(c,db,B) \\ \lor \\ \exists B\ (\mathit{project}(B) \land \mathit{inArea}(B,db)) \end{array}}$$

- Consider the database $D = \{\mathit{project}(a), \mathit{inArea}(a,db)\}$
- 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 $\mathit{hasCollaborator}(t,db,t)$ during the chase

Unsound Rewritings

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

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

$$Q_\Sigma = \exists B\ \mathit{hasCollaborator}(c,db,B)$$
$$\vee$$
$$\exists B\ (\mathit{project}(B) \wedge \mathit{inArea}(B,db))$$

the fact that B in the original query participates in a join is lost after the application of the rewriting step since B is unified with an ∃-variable

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

Incomplete Rewritings

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

$Q = \exists A \exists B \exists C\ (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A))$

$$\begin{aligned} Q_\Sigma = {} & \exists A \exists B \exists C\ (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A)) \\ & \vee \\ & \exists A \exists B \exists C \exists E \exists F\ (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{hasCollaborator}(A,E,F)) \end{aligned}$$

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

Incomplete Rewritings

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

$\forall X \forall Y \forall Z\ (\mathit{hasCollaborator}(X,Y,Z) \rightarrow \mathit{collaborator}(X))\}$

$Q = \exists A \exists B \exists C\ (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A))$

$$Q_\Sigma = \exists A \exists B \exists C\ (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A))$$

$$\vee$$

$$\exists A \exists B \exists C \exists E \exists F\ (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{hasCollaborator}(A,E,F))$$

$$\vee$$

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

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

Minimization Step

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

$\forall X \forall Y \forall Z\, (\mathit{hasCollaborator}(X,Y,Z) \rightarrow \mathit{collaborator}(X))\}$

$Q = \exists A \exists B \exists C\, (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A))$

$$Q_\Sigma = \exists A \exists B \exists C\, (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A))$$

$$\vee$$

$$\exists A \exists B \exists C \exists E \exists F\, (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{hasCollaborator}(A,E,F))$$

$$\mathit{hasCollaborator}(A,B,C)$$

Minimization Step

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

$\forall X \forall Y \forall Z\, (\mathit{hasCollaborator}(X,Y,Z) \rightarrow \mathit{collaborator}(X))\}$

$Q = \exists A \exists B \exists C\, (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A))$

$Q_\Sigma = \exists A \exists B \exists C\, (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A))$

$\vee$

$\exists A \exists B \exists C \exists E \exists F\, (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{hasCollaborator}(A,E,F))$

$\vee$

$\exists A \exists B \exists C\, (\mathit{hasCollaborator}(A,B,C))$ - by minimization

Minimization Step

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

$\forall X \forall Y \forall Z\ (\mathit{hasCollaborator}(X,Y,Z) \rightarrow \mathit{collaborator}(X))\}$

$Q = \exists A \exists B \exists C\ (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A))$

$Q_\Sigma = \exists A \exists B \exists C\ (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{collaborator}(A))$

$\vee$

$\exists A \exists B \exists C \exists E \exists F\ (\mathit{hasCollaborator}(A,B,C) \wedge \mathit{hasCollaborator}(A,E,F))$

$\vee$

$\exists A \exists B \exists C\ (\mathit{hasCollaborator}(A,B,C))$ - by minimization

$\vee$

$\exists B \exists C\ (\mathit{project}(C) \wedge \mathit{inArea}(C,B))$ - by rewriting

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);   // 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);   // 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_Σ ∨ 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**

Minimization Step Revisited

$\Sigma = \{\forall X\, (P(X) \rightarrow \exists Y\, R(X,Y))\}$

$Q = \exists A_1 \ldots \exists A_n \exists B\, (S_1(A_1) \wedge R(A_1,B) \wedge \ldots \wedge S_n(A_n) \wedge R(A_n,B))$

exponentially many minimization steps must be applied in order to get the query

$$\exists A \exists B\, (S_1(A) \wedge \ldots \wedge S_n(A) \wedge R(A,B))$$

and then apply the rewriting step, which will lead to the query

$$\exists A\, (S_1(A) \wedge \ldots \wedge S_n(A) \wedge P(A))$$

Minimization Step Revisited

Σ = $\{\forall X\, (P(X) \rightarrow \exists Y\, R(X,Y))\}$

Q = $\exists A_1 \ldots \exists A_n \exists B\, (S_1(A_1) \wedge R(A_1,B) \wedge \ldots \wedge S_n(A_n) \wedge R(A_n,B))$

Piece-based Rewriting

- Instead of rewriting a single atom
- Rewrite a set of atoms that have to be rewritten together

Computing the Piece

Input: CQ q, atom $\alpha = R(t_1,\ldots,t_n)$ in q, rule σ

Output: piece of α in q w.r.t. σ

Piece := $\{R(t_1,\ldots,t_n)\}$;

while TRUE do

  if *Piece* and head(σ) do not unify then

    return Ø;

  h := most general unifier of *Piece* and head(σ);

  if h violates points 2 or 4 of the applicability condition then

    return Ø;

  if h violates point 3 of the applicability condition then

    Piece := *Piece* ∪ {atoms containing a variable that unifies with an ∃-variable};

  else

    return *Piece*;

The Piece-based Rewriting Algorithm

$Q_\Sigma := Q$;

repeat

$Q_{aux} := Q_\Sigma$;

foreach disjunct q of Q_{aux} do

foreach atom α in q do

foreach rule σ in Σ do

//Rewriting Step

if σ is <u>applicable</u> to α then

$q_{rew} := \mathit{rewrite}(q,\alpha,\sigma)$; // resolve α using σ

if q_{rew} does not appear in Q_Σ (modulo variable renaming) then

$Q_\Sigma := Q_\Sigma \vee q_{rew}$;

//Minimization Step

P := piece of α in q w.r.t. σ;

$q_{min} := \mathit{minimize}(q,\mathrm{P})$; // apply the most general unifier of P on q

if q_{min} does not appear in Q_Σ (modulo variable renaming) then

$Q_\Sigma := Q_\Sigma \vee q_{min}$;

until $Q_{aux} = Q_\Sigma$;

return Q_Σ;

Termination

$\Sigma = \{\forall X \forall Y\, (R(X,Y) \wedge P(Y) \rightarrow P(X))\}$

$Q = \exists X\, P(X)$

$$Q_\Sigma = \exists X\, P(X)$$
$$\vee$$
$$\exists X \exists Y_1\, (R(c,Y_1) \wedge P(Y_1))$$
$$\vee$$
$$\exists X \exists Y_1 \exists Y_2\, (R(c,Y_1) \wedge R(Y_1,Y_2) \wedge P(Y_2))$$
$$\vee$$
$$\exists X \exists Y_1 \exists Y_2 \exists Y_3\, (R(c,Y_1) \wedge R(Y_1,Y_2) \wedge R(Y_2,Y_3) \wedge P(Y_3))$$
$$\vee$$
$$\ldots$$

- The piece-based rewriting algorithm does not terminate
- However, there exists a finite UCQ-rewritings, that is, $\exists X\, P(X)$

...careful application of the homomorphism check

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