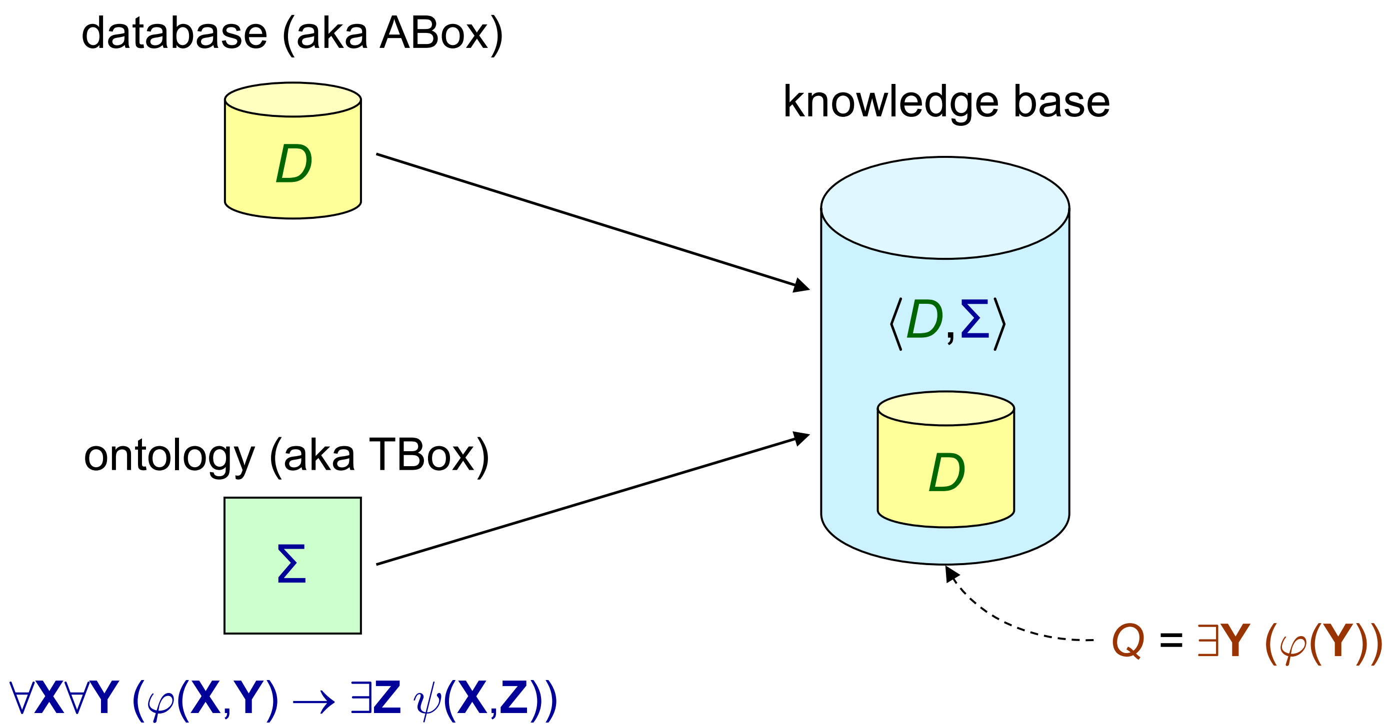
Sebastian Rudolph
International Center for Computational Logic
TU Dresden

Existential Rules – Lecture 7

Adapted from slides by Andreas Pieris and Michaël Thomazo
Winter Term 2025/26

BCQ-Answering: Our Main Decision Problem



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

Termination of the Chase

- Drop the existential quantification
  - We obtain the class of full existential rules
  - Very close to Datalog ✓

- Drop the recursive definitions
  - We obtain the class of acyclic existential rules
  - A.k.a. non-recursive existential rules ✓

Sum Up

	Data Complexity	
FULL	PTIME-c	Naïve algorithm
		Reduction from Monotone Circuit Value problem
ACYCLIC	in LOGSPACE	Not covered here

	Combined Complexity	
FULL	EXPTIME-c	Naïve algorithm
		Simulation of a deterministic exponential time TM
ACYCLIC	NEXPTIME-c	Small witness property
		Reduction from Tiling problem

Recall our Example

D

person(Alice)

Σ

$\forall X\ (Person(X) \rightarrow \exists Y\ (hasParent(X,Y) \wedge Person(Y)))$

$\text{chase}(D,\Sigma) = D \cup \{hasParent(\text{Alice}, z_1), Person(z_1),$

$hasParent(z_1, z_2), Person(z_2),$

$hasParent(z_2, z_3), Person(z_3), \ldots$

Existential quantification & recursive definitions

are key features for modelling ontologies

Linear Existential Rules

- A linear existential rule is an existential rule of the form

$$\forall X \forall Y\ (P(X,Y) \rightarrow \exists Z\ \psi(X,Z))$$

  where $P(X,Y)$ is an atom (which is trivially a guard)

- We denote LINEAR the class of linear existential rules

- A local property - we can inspect one rule at a time
  - $\Rightarrow$ given Σ, we can decide in linear time whether $\Sigma \in$ LINEAR
  - $\Rightarrow \Sigma_1 \in$ LINEAR, $\Sigma_2 \in$ LINEAR $\Rightarrow (\Sigma_1 \cup \Sigma_2) \in$ LINEAR

- Strictly more expressive than DL-Lite

- Infinite chase - $\forall X\ (\mathit{Person}(X) \rightarrow \exists Y\ (\mathit{hasParent}(X,Y) \wedge \mathit{Person}(Y)))$

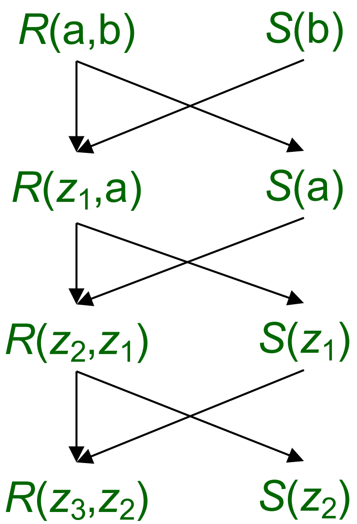
- But, BCQ-Answering is decidable - the chase has finite treewidth

Chase Graph

The chase can be naturally seen as a graph - chase graph

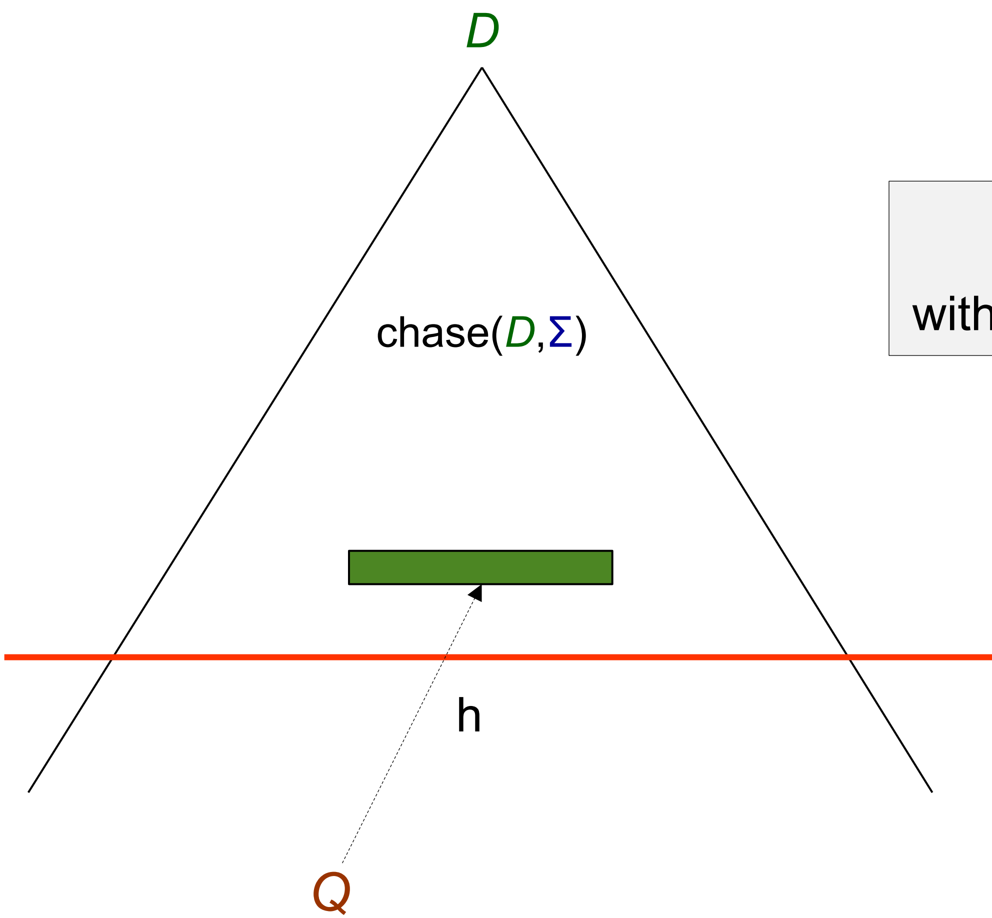
$D = \{R(a,b), S(b)\}$

$$\Sigma = \begin{cases} \forall X \forall Y\ (R(X,Y) \wedge S(Y) \rightarrow \exists Z\ R(Z,X)) \\ \forall X \forall Y\ (R(X,Y) \rightarrow S(X)) \end{cases}$$



For LINEAR, the chase graph is a forest

Bounded Derivation-Depth Property



For LINEAR, $k = |Q| \cdot m$

with $m = |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$

depth k

k does not depend on D

chase graph up to depth k

$$\text{chase}(D,\Sigma) \models Q \quad \Rightarrow \quad \text{chase}^k(D,\Sigma) \models Q$$

Combined Complexity of LINEAR

Theorem: BCQ-Answering under LINEAR is in PSPACE w.r.t. the combined complexity

Proof (cont.):

At each step we need to maintain

- $O(|Q|)$ atoms
- A counter $ctr \leq (|Q|)^2 \cdot |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$

- Thus, we need polynomial space

- The claim follows since NPSPACE = PSPACE

Combined Complexity of LINEAR

Theorem: BCQ-Answering under LINEAR is in PSPACE w.r.t. the combined complexity

We cannot do better:

Theorem: BCQ-Answering under LINEAR is PSPACE-hard w.r.t. the combined complexity

Proof : By simulating a deterministic polynomial space Turing machine

PSPACE-hardness of LINEAR

Our Goal: Encode the polynomial space computation of a DTM M on input string I using a database D, a set $\Sigma \in$ LINEAR, and a BCQ Q such that

$D \wedge \Sigma \models Q$ iff M accepts I using at most $n = (|I|)^k$ cells

PSPACE-hardness of LINEAR

- Assume that the tape alphabet is $\{0,1,\sqcup\}$
- Suppose that M halts on $I = \alpha_1 \ldots \alpha_m$ using $n = m^k$ cells, for $k > 0$

Initial configuration - the database D

$$Config(s_{init},\alpha_1,\ldots,\alpha_m,\underbrace{\sqcup,\ldots,\sqcup}_{n-m},1,\underbrace{0,\ldots,0}_{n-1})$$

PSPACE-hardness of LINEAR

- Assume that the tape alphabet is $\{0,1,\sqcup\}$
- Suppose that M halts on $I = \alpha_1 \ldots \alpha_m$ using $n = m^k$ cells, for $k > 0$

Transition rule - $\delta(s_1,\alpha) = (s_2,\beta,+1)$

for each $i \in \{1,\ldots,n\}$:

$$\forall X\, (\mathit{Config}(s_1,X_1,\ldots,X_{i-1},\alpha,X_{i+1},\ldots,X_n,\overbrace{0,\ldots,0}^{i-1},1,\overbrace{0,\ldots,0}^{n-i}) \rightarrow$$

$$\mathit{Config}(s_2,X_1,\ldots,X_{i-1},\beta, X_{i+1},\ldots,X_n,\underbrace{0,\ldots,0}_{i},1,\underbrace{0,\ldots,0}_{n-i-1}))$$

PSPACE-hardness of LINEAR

- Assume that the tape alphabet is $\{0,1,\sqcup\}$
- Suppose that M halts on $I = \alpha_1 \dots \alpha_m$ using $n = m^k$ cells, for $k > 0$

$$D \wedge \Sigma \models \exists X\, Config(s_{acc}, X) \quad \text{iff} \quad M \text{ accepts } I$$

…but, the rules are not constant-free

we can eliminate the constants by applying a simple trick

PSPACE-hardness of LINEAR

Initial configuration - the database *D*

auxiliary constants for the states

and the tape alphabet

$$\mathit{Config}(s_{init},\alpha_1,\ldots,\alpha_m,\underbrace{\sqcup,\ldots,\sqcup}_{n-m},1,\underbrace{0,\ldots,0}_{n-1},s_1,\ldots s_\ell,0,1,\sqcup)$$

PSPACE-hardness of LINEAR

Transition rule - $\delta(s_1,0) = (s_2,\sqcup,+1)$

for each $i \in \{1,\ldots,n\}$:

$$Config(S_1,X_1,\ldots,X_{i-1},Z,X_{i+1},\ldots,X_n,\underbrace{Z,\ldots,Z}_{i-1},O,\underbrace{Z,\ldots,Z}_{n-i},S_1,\ldots S_\ell,Z,O,B) \to$$

$$Config(S_2,X_1,\ldots,X_{i-1},B, X_{i+1},\ldots,X_n,\underbrace{Z,\ldots,Z}_{i},O,\underbrace{Z,\ldots,Z}_{n-i-1}, S_1,\ldots S_\ell,Z,O,B)$$

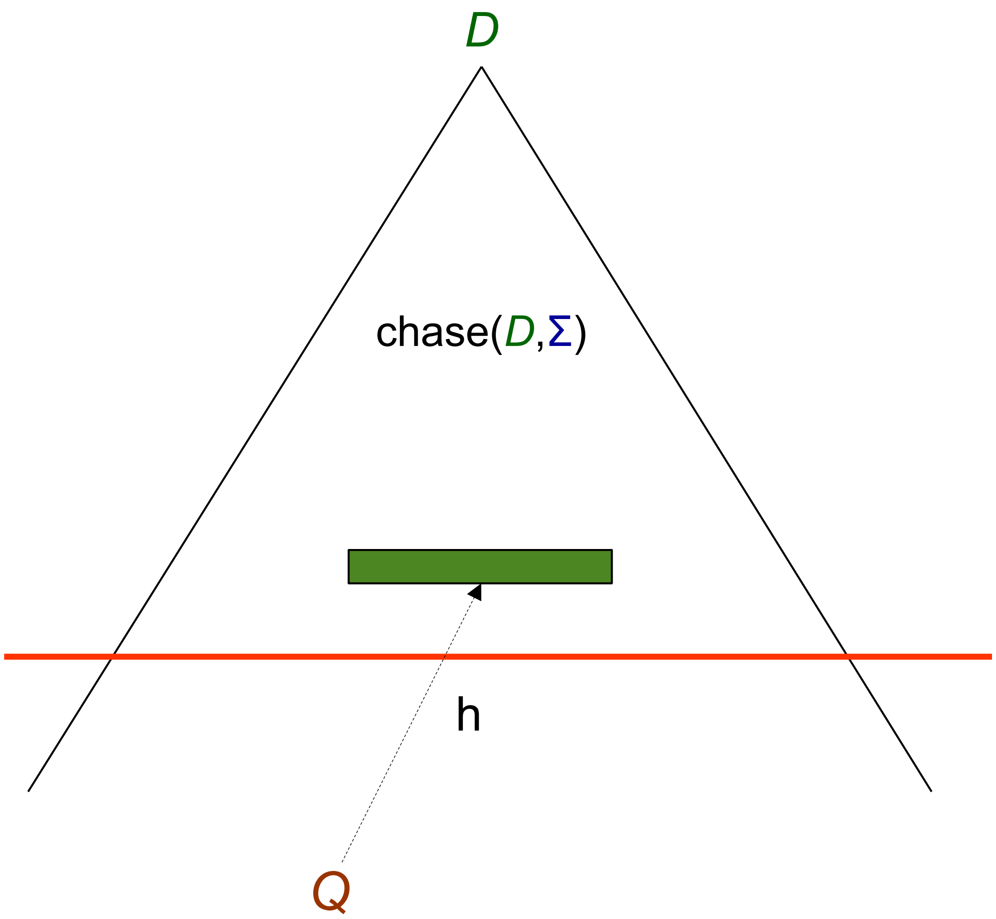
(∀-quantifiers are omitted)

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

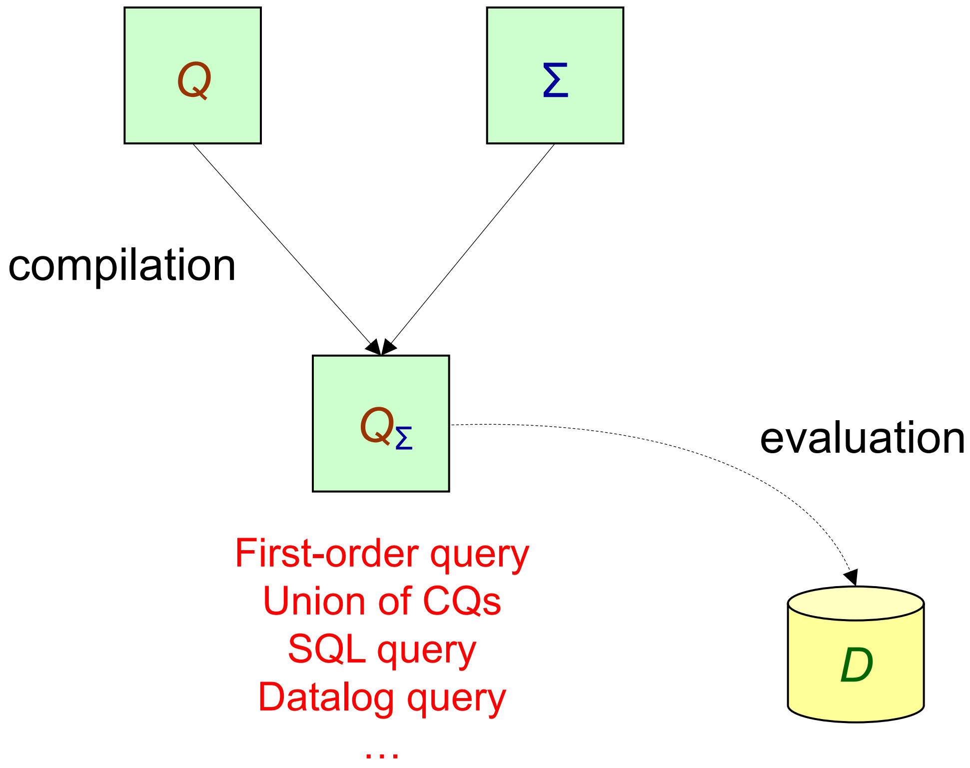
Forward Chaining Techniques



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

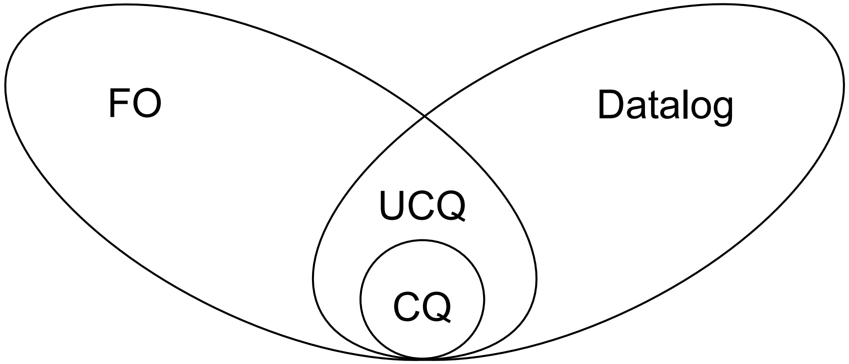
Issues in Query Rewriting

- 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?
- …

the above issues, and more, will be covered next…

Target Query Language

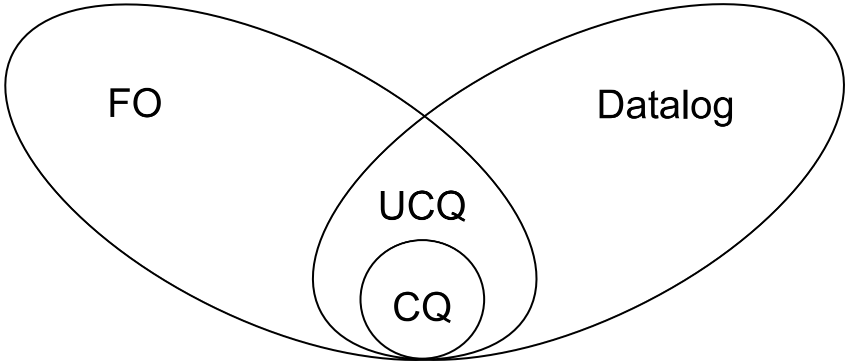
we target the weakest query language



	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

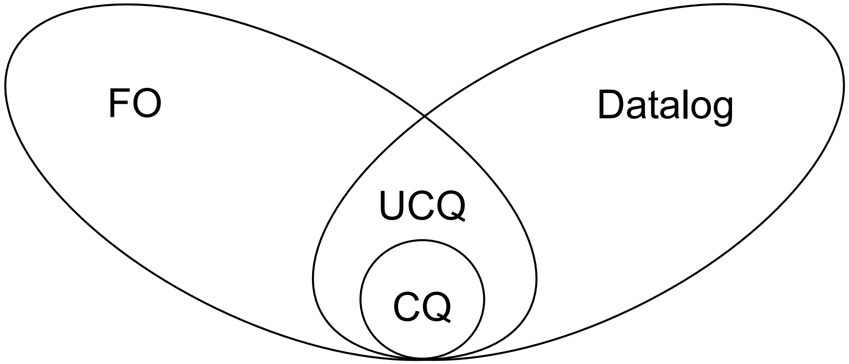
Theorem: BCQ-Answering under $\mathcal{L}$, where $\mathcal{L} \in$ {FULL, ACYCLIC, LINEAR}, is not CQ-rewritable

Proof:

- It suffices to construct a set $\Sigma \in \mathcal{L}$ and a CQ Q for which the following holds: there is no CQ Q_Σ such that for every database D, $D \wedge \Sigma \vDash Q$ iff $D \vDash Q_\Sigma$
- Let $\Sigma = \{\forall X\, (P(X) \rightarrow S(X))\}$ and $Q = S(a)$
- Clearly, for every database D, $D \wedge \Sigma \vDash S(a)$ iff $D \vDash P(a) \vee S(a)$
- Assume there exists a CQ-rewriting Q_Σ
- Since $P(a) \vee S(a)$ is a rewriting, $P(a) \rightarrow Q_\Sigma$ or $S(a) \rightarrow Q_\Sigma$ ($\rightarrow$ denotes the existence of a homomorphism)
- Moreover, since Q_Σ is a rewriting, $Q_\Sigma \rightarrow P(a)$ and $Q_\Sigma \rightarrow S(a)$
- Therefore, $S(a) \rightarrow P(a)$ or $P(a) \rightarrow S(a)$, which is a contradiction

Target Query Language

we target the weakest query language



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

Union of Conjunctive Queries (UCQ)

A union of conjunctive queries (UCQ) is an expression

$$\exists Y\ (\varphi_1(X,Y)) \vee \ldots \vee \exists Y\ (\varphi_n(X,Y))$$

- X and Y are tuples of variables of V
- $\varphi_k(X,Y)$ is a conjunctive query

Union of Conjunctive Queries (UCQ)

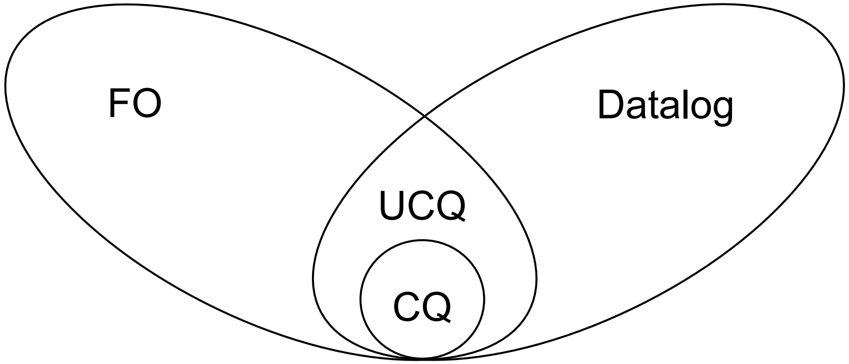
A union of conjunctive queries (UCQ) is an expression

$$\underbrace{\exists Y\,(\varphi_1(X,Y))}_{Q_1} \vee \ldots \vee \underbrace{\exists Y\,(\varphi_n(X,Y))}_{Q_n}$$

$$Q(J) \;=\; \cup_{k \in \{1,\ldots,n\}}\; Q_k(J)$$

Target Query Language

we target the weakest query language



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

Target Query Language

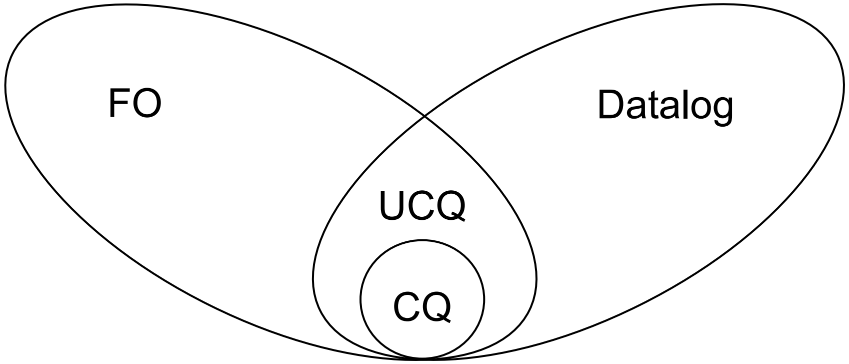
$\Sigma = \{\forall X\, (P(X) \rightarrow T(X)), \forall X \forall Y\, (R(X,Y) \rightarrow S(X))\}$

$Q = \exists X \exists Y\, (S(X) \wedge U(X,Y) \wedge T(Y))$

$$Q_\Sigma = \exists X \exists Y\, (S(X) \wedge U(X,Y) \wedge T(Y))$$
$$\vee$$
$$\exists X \exists Y\, (S(X) \wedge U(X,Y) \wedge P(Y))$$
$$\vee$$
$$\exists X \exists Y \exists Z\, (R(X,Z) \wedge U(X,Y) \wedge T(Y))$$
$$\vee$$
$$\exists X \exists Y \exists Z\, (R(X,Z) \wedge U(X,Y) \wedge P(Y))$$

Target Query Language

we target the weakest query language



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

Target Query Language

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

$Q = P(c)$

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

- This cannot be written as a finite UCQ (or even FO query)
- It can be written as $\exists X \exists Y\, (R(c,X) \wedge R^*(X,Y) \wedge P(Y))$, but transitive closure is not FO-expressible

Target Query Language

Theorem: BCQ-Answering under FULL is not UCQ-rewritable

Proof 1:

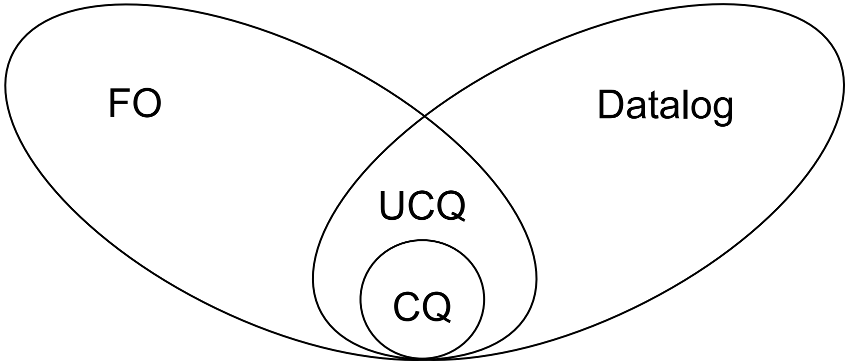
- Transitive closure is not FO-expressible

Proof 2:

- Via a complexity-theoretic argument
- Assume that BCQ-Answering under FULL is UCQ-rewritable
- Thus, BCQ-Answering under FULL is in AC_0 w.r.t. to the data complexity
- BCQ-Answering under FULL is PTIME-hard w.r.t. to the data complexity
- Therefore, AC_0 = PTIME which is a contradiction

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

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

$$\downarrow$$

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

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

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

$$\ldots$$

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

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

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

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

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

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

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

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

$$\vee$$

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

Unsound Rewritings

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

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

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

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

After applying the rewriting step we obtain the following UCQ

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

$$\lor$$

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

Unsound Rewritings

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

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

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

- Consider the database $D = \{\mathit{project}(a), \mathit{inArea}(a,db)\}$
- Clearly, $D \models Q_\Sigma$
- 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

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

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

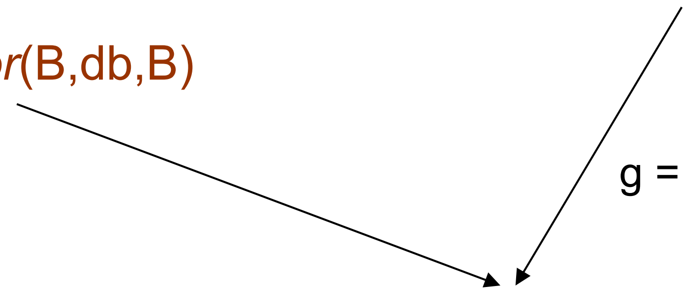
$$Q_{\Sigma} = \exists B\ \mathit{hasCollaborator}(c,db,B)$$
$$\vee$$
$$\exists B\ (\mathit{project}(B) \wedge \mathit{inArea}(B,db))$$

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

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

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

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

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

- 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*(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 $h : \text{terms}(\text{head}(\sigma)) \cup \text{terms}(\alpha) \rightarrow \text{terms}(\alpha)$

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 is crucial for soundness, may destroy completeness

Incomplete Rewritings

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

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

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

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

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

Incomplete Rewritings

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

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

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

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

$\vee$

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

$\vee$

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

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

Minimization Step

Σ = {∀X∀Y (*project*(X) ∧ *inArea*(X,Y) → ∃Z *hasCollaborator*(Z,Y,X)),
∀X∀Y∀Z (*hasCollaborator*(X,Y,Z) → *collaborator*(X))}

Q = ∃A∃B∃C (*hasCollaborator*(A,B,C) ∧ *collaborator*(A))

Q_Σ = ∃A∃B∃C (*hasCollaborator*(A,B,C) ∧ *collaborator*(A))

∨

∃A∃B∃C∃E∃F (*hasCollaborator*(A,B,C) ∧ *hasCollaborator*(A,E,F))

hasCollaborator(A,B,C)

Minimization Step

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 = $\exists A \exists B \exists C$ (*hasCollaborator*(A,B,C) $\wedge$ *collaborator*(A))

Q_Σ = $\exists A \exists B \exists C$ (*hasCollaborator*(A,B,C) $\wedge$ *collaborator*(A))

$\vee$

$\exists A \exists B \exists C \exists E \exists F$ (*hasCollaborator*(A,B,C) $\wedge$ *hasCollaborator*(A,E,F))

$\vee$

$\exists A \exists B \exists C$ (*hasCollaborator*(A,B,C)) - by minimization

Minimization Step

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

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

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

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

$\vee$

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

$\vee$

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

$\vee$

$\exists B \exists C\, (project(C) \wedge inArea(C,B))$ - 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 <u>applicable</u> 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 := $Q_\Sigma \vee q_{rew}$;
  //Minimization Step
    foreach pair of atoms α,β in q that <u>unify</u> 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 := $Q_\Sigma \vee q_{min}$;
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!!!

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

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

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

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

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

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

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

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

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

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 $\alpha = R(t_1,...,t_n)$ in q, rule σ

Output: piece of α in q w.r.t. σ

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

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_Σ := Q;
repeat
  Q_{aux} := Q_Σ;
  foreach disjunct q of Q_{aux} do
    foreach atom α in q do
      foreach rule σ in Σ do
        //Rewriting Step
        if σ is applicable to α then
          q_{rew} := $rewrite(q,\alpha,\sigma)$; //we 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
        P := piece of α in q w.r.t. σ;
        q_{min} := $minimize(q,P)$; //we apply the MGU of P on q
        if q_{min} does not appear in Q_Σ (modulo variable renaming) then
          Q_Σ := $Q_\Sigma \vee q_{min}$;
until $Q_{aux} = Q_\Sigma$;
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 homomorohism 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

$\Rightarrow$ a more refined approach is needed