# Ontology-Based Query Answering: The Basics

# Outline

# References


- Meghyn Bienvenu, Magdalena Ortiz: Ontology-Mediated Query Answering with Data-Tractable Description Logics. Reasoning Web 2015: 218-307
- Marie-Laure Mugnier, Michaël Thomazo: An Introduction to Ontology-Based Query Answering with Existential Rules. Reasoning Web 2014: 245-278
- Andreas Pieris, Michaël Thomazo: Ontological Conjunctive Query Answering via Existential Rules. ESSLI 2015

  Not available online, but some slides come from this tutorial.
- Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Eric Salvat: On rules with existential variables: Walking the decidability line. Artif. Intell. 175(9-10): 1620-1654 (2011)
- Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao: Ontop: Answering SPARQL queries over relational databases. Semantic Web 8(3): 471-487 (2017)

# Motivation for Ontology-Based Data Access

"Data is stored in various heterogeneous formats over many differently structured databases. As a result, the gathering of only relevant data spread over disparate sources becomes a very time consuming task."

Jim Crompton, W3C Workshop on Semantic Web in Oil & Gas Industry, 2008

# The Challenge of Accessing Big Data

## The Statoil Example

*Experts in geology and geophysics develop stratigraphic models of unexplored areas on the basis of data acquired from previous operations at nearby geographical locations.*

Fact:

- 1000TB of relational data
- Using diverse schemata
- spread over 2000 tables, over mutiple individual databases

Data Access for Exploration::

- 900 experts in Statoil Exploration
- Up to 4 days for new data access queries – assistance by IT experts
- 30 - 70 % of time spent on data gathering

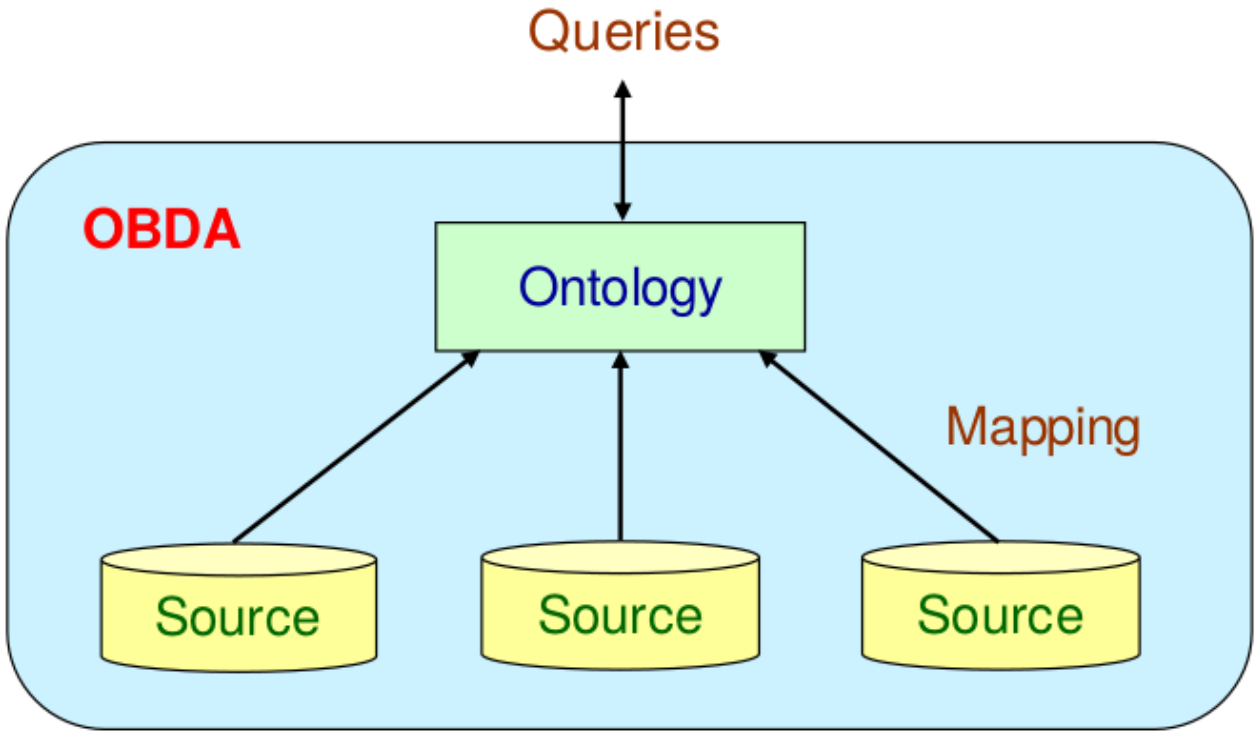
# Ontology-Based Data Access (OBDA)

- achieve transparency in accessing data using logic
- manage data by exploiting knowledge representation techniques

Key principles underlying OBDA:

- conceptual, high level representation of the domain of interest in terms of an *ontology*
- map the data sources to the ontology – do not migrate the data
- specify all information requests to the data in terms of the ontology

# Ontology-Based Data Access: Architecture




- Ontology: provides a unified conceptual view of the data
- Data Sources: external and independent (possibly multiple and heterogeneous)
- Mappings: semantically link data at the sources with the ontology

# Ontology-Based Data Access: An Example

Ontology – logical representation of the domain of interest

$$\forall X(\text{Researcher}(X) \rightarrow \exists Y(\text{worksFor}(X, Y) \wedge \text{Project}(Y)))$$
$$\forall X(\text{Project}(X) \rightarrow \exists Y(\text{worksFor}(Y, X) \wedge \text{Researcher}(Y)))$$
$$\forall X, Y(\text{worksFor}(X, Y) \rightarrow \text{Researcher}(X) \wedge \text{Project}(Y))$$
$$\forall X(\text{Project}(\text{X}) \rightarrow \exists Y(\text{PrName}(X, Y)))$$

# Ontology-Based Data Access: An Example

Relational Database $D$ – a single database that represents the sources

worksIn

| SSN | Name |
|---|---|
| 100 | AAA |
| 200 | BBB |
| 300 | CCC |

Intuitively, represents "The researcher with SSN 100 works for project AAA".

# Ontology-Based Data Access: An Example

Mappings $M$ – semantically link data at the sources with the ontology

$$\begin{array}{lcl} & & \mathrm{Researcher}(\mathrm{person}(SSN)) \wedge \\ \text{SELECT SSN, Name} & \rightsquigarrow & \mathrm{Project}(\mathrm{proj}(\text{Name})) \wedge \\ \text{FROM } \mathrm{worksIn} & & \mathrm{worksFor}(\mathrm{person}(\text{SSN}), \mathrm{proj}(\text{Name})) \wedge \\ & & \mathrm{PrName}(\mathrm{proj}(\text{Name}), \text{Name}) \end{array}$$

- $\mathrm{person}$ and $\mathrm{proj}$ are constructors to create objects of the ontology from tuple of values from the database
- they are Skolem functions

Mappings could have varying expressivity, including negation, inequality,...

# Ontology-Based Data Access: An Example

worksIn

| SSN | Name |
|---|---|
| 100 | AAA |
| 200 | BBB |
| 300 | CCC |

SELECT SSN, Name
FROM worksIn

$\rightsquigarrow$

$$\begin{aligned}&\text{Researcher}(\text{person}(\textit{SSN}))\wedge\\&\text{Project}(\text{proj}(\text{Name}))\ \wedge\\&\text{worksFor}(\text{person}(\text{SSN}),\text{proj}(\text{Name}))\wedge\\&\text{PrName}(\text{proj}(\text{Name}),\text{Name})\end{aligned}$$

# Ontology-Based Data Access: An Example

worksIn

| SSN | Name |
|---|---|
| 100 | AAA |
| 200 | BBB |
| 300 | CCC |

SELECT SSN, Name
FROM worksIn

$\rightsquigarrow$

$\text{Researcher}(\text{person}(\mathit{SSN})) \wedge$
$\text{Project}(\text{proj}(\text{Name})) \wedge$
$\text{worksFor}(\text{person}(\text{SSN}), \text{proj}(\text{Name})) \wedge$
$\text{PrName}(\text{proj}(\text{Name}), \text{Name})$

$$\text{Researcher}(\text{person}(100)) \wedge \text{Project}(\text{proj}(\mathit{AAA})) \wedge$$
$$\text{worksFor}(\text{person}(100), \text{proj}(\mathit{AAA})) \wedge \text{PrName}(\text{proj}(\mathit{AAA}), \mathit{AAA})$$

# Ontology-Based Data Access: An Example

worksIn

| SSN | Name |
|---|---|
| 100 | AAA |
| 200 | BBB |
| 300 | CCC |

SELECT SSN, Name
FROM worksIn

$\rightsquigarrow$

$\text{Researcher}(\text{person}(SSN)) \wedge$
$\text{Project}(\text{proj}(\text{Name})) \wedge$
$\text{worksFor}(\text{person}(\text{SSN}), \text{proj}(\text{Name})) \wedge$
$\text{PrName}(\text{proj}(\text{Name}),\text{Name})$

$$\text{Researcher}(\text{person}(100)) \wedge \text{Project}(\text{proj}(AAA)) \wedge$$
$$\text{worksFor}(\text{person}(100), \text{proj}(AAA)) \wedge \text{PrName}(\text{proj}(AAA), AAA)$$

$$\text{Researcher}(\text{person}(200)) \wedge \text{Project}(\text{proj}(BBB)) \wedge$$
$$\text{worksFor}(\text{person}(200), \text{proj}(BBB)) \wedge \text{PrName}(\text{proj}(BBB), BBB)$$

# Ontology-Based Data Access: An Example

worksIn

| SSN | Name |
|---|---|
| 100 | AAA |
| 200 | BBB |
| 300 | CCC |

SELECT SSN, Name
FROM worksIn

$\rightsquigarrow$

$\text{Researcher}(\text{person}(SSN)) \wedge$
$\text{Project}(\text{proj}(\text{Name})) \wedge$
$\text{worksFor}(\text{person}(\text{SSN}), \text{proj}(\text{Name})) \wedge$
$\text{PrName}(\text{proj}(\text{Name}), \text{Name})$

$$\text{Researcher}(\text{person}(100)) \wedge \text{Project}(\text{proj}(AAA)) \wedge$$
$$\text{worksFor}(\text{person}(100), \text{proj}(AAA)) \wedge \text{PrName}(\text{proj}(AAA), AAA)$$

$$\text{Researcher}(\text{person}(200)) \wedge \text{Project}(\text{proj}(BBB)) \wedge$$
$$\text{worksFor}(\text{person}(200), \text{proj}(BBB)) \wedge \text{PrName}(\text{proj}(BBB), BBB)$$

$$\text{Researcher}(\text{person}(300)) \wedge \text{Project}(\text{proj}(CCC)) \wedge$$
$$\text{worksFor}(\text{person}(300), \text{proj}(CCC)) \wedge \text{PrName}(\text{proj}(CCC), CCC)$$

# Ontology-Based Data Access: An Example

$$\forall X(\text{Researcher}(X) \rightarrow \exists Y(\text{worksFor}(X, Y) \wedge \text{Project}(Y)))$$
$$\forall X(\text{Project}(X) \rightarrow \exists Y(\text{worksFor}(Y, X) \wedge \text{Researcher}(Y)))$$
$$\forall X, Y(\text{worksFor}(X, Y) \rightarrow \text{Researcher}(X) \wedge \text{Project}(Y)$$
$$\forall X(\text{Project(X)} \rightarrow \exists Y(\text{PrName}(X, Y)))$$

$$\text{Researcher}(\text{person}(100)) \wedge \text{Project}(\text{proj}(AAA)) \wedge$$
$$\text{worksFor}(\text{person}(100), \text{proj}(AAA)) \wedge \text{PrName}(\text{proj}(AAA), AAA)$$
$$\text{Researcher}(\text{person}(200)) \wedge \text{Project}(\text{proj}(BBB)) \wedge$$
$$\text{worksFor}(\text{person}(200), \text{proj}(BBB)) \wedge \text{PrName}(\text{proj}(BBB), BBB)$$
$$\text{Researcher}(\text{person}(300)) \wedge \text{Project}(\text{proj}(CCC)) \wedge$$
$$\text{worksFor}(\text{person}(300), \text{proj}(CCC)) \wedge \text{PrName}(\text{proj}(CCC), CCC)$$

# Ontology-Based Query Answering

- we assume a unique database source
- whose vocabulary coincide with the vocabulary of the ontology

No more mappings! We focus on the core problem of query answering while taking an ontology into account. The problem boils down to decide, given a database $D$, an ontology $\Sigma$ and a query $q$ whether:

$$D, \Sigma \models q$$

# Influence of Ontology and Query Languages

**What is the/a right ontology language?**

- there is a wide spectrum of languages that differ in expressive power and computational complexity
- an important aspect is the scalability to large amounts of data

**What is the/a right query language?**

- "core" fragment of traditional database query languages?
- navigational components, such as can be seen in SPARQL?
- counting aspects?

# Combined vs. Data Complexity

When considering OBQA, there are two classical ways of measuring complexity:

- combined complexity, where the database, the ontology and the query are considered part of the input;
- data complexity, where only the database is considered part of the input, whereas the ontology and the query are considered to be constant.

# Query Languages

Several query languages have been introduced/studied so far, among which:

- conjunctive queries;
- union of conjunctive queries;
- first-order queries;
- (conjunctive) regular path queries;
- Datalog.

We will mostly focus on conjunctive queries in this course.

# Conjunctive Queries – Syntax

A conjunctive query (CQ) is an expression of the shape

$$\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}),$$

where:

- $\mathbf{X}$ and $\mathbf{Y}$ are tuples of variables
- $\varphi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms whose set of variables is $\mathbf{X} \cup \mathbf{Y}$, and possibly containing constants

Forms the SELECT-FROM-WHERE fragment of SQL. Core of numerous query languages.

$$q(x) :- \exists y \ \exists z \ P(x, y) \wedge S(x, z) \wedge T(y, z)$$

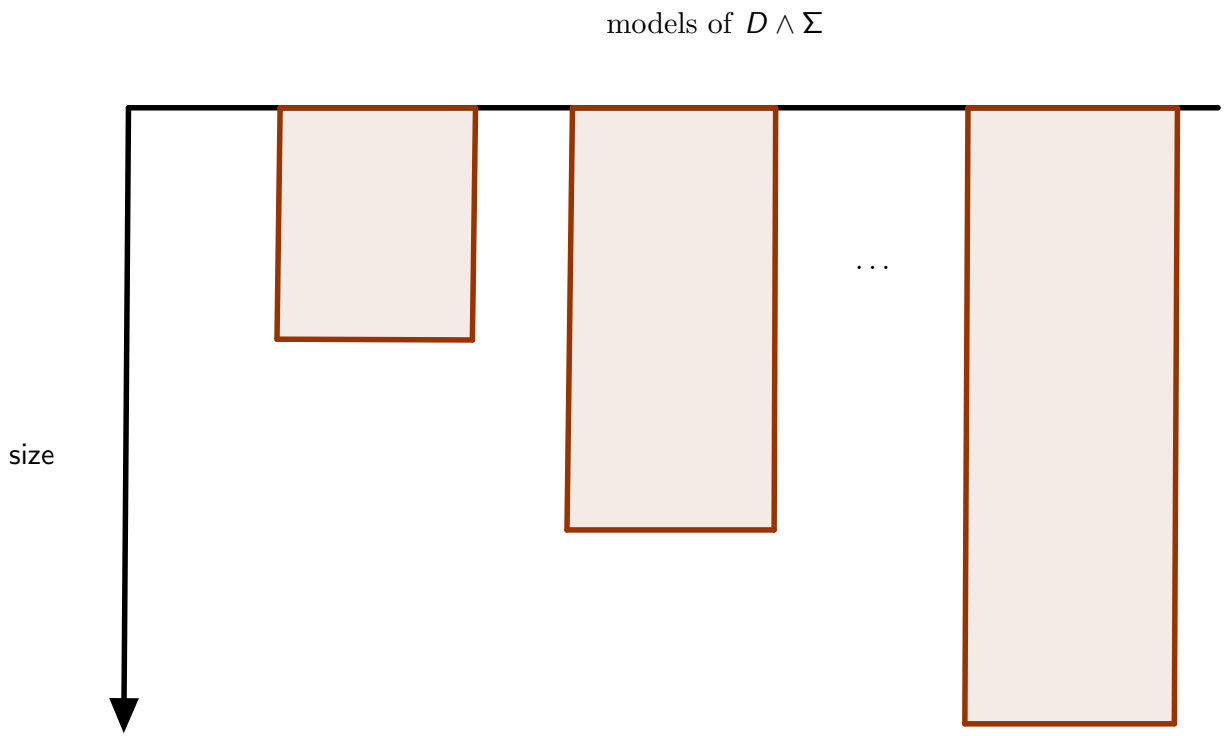
# Conjunctive Queries – Semantics

- a **match** of a CQ $\exists \mathbf{Y}\ \varphi(\mathbf{X}, \mathbf{Y})$ in a database $D$ is a homomorphism $h$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$ – all the atoms of the query are satisfied
- the **answer** to $Q = \exists \mathbf{Y}\ \varphi(\mathbf{X}, \mathbf{Y})$ over $D$ is the set of tuples

$$Q(D) = \{h(\mathbf{X}) \mid h \text{ is a match of } Q \text{ in } D\}$$

NB: a tuple in $Q(D)$ is commonly called an answer of $Q$ over $D$.

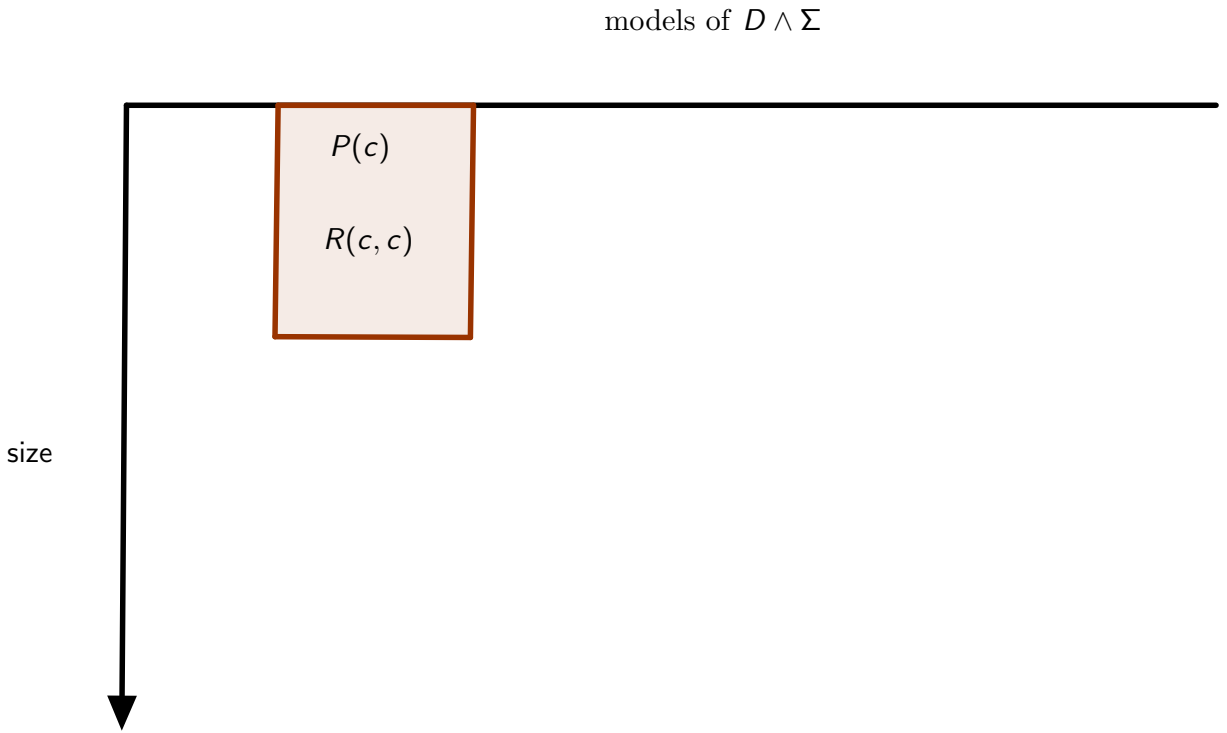
# Two Dimensions of Infinity




$D \wedge \Sigma$ admits infinitely many models, and each one may be of of infinite size.

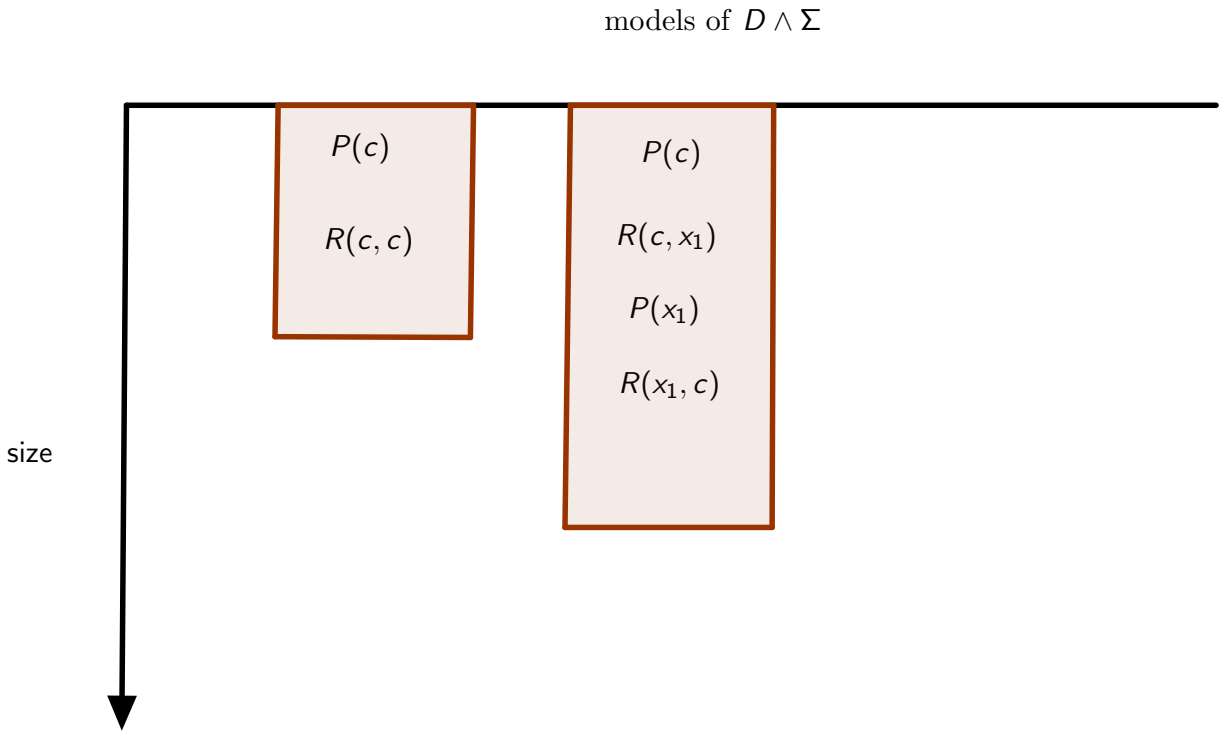
# Two Dimensions of Infinity

Consider $D = \{P(c)\}$ and
$\Sigma = \{\forall X(P(X) \rightarrow \exists Y(R(X, Y) \wedge P(Y)))\}$.



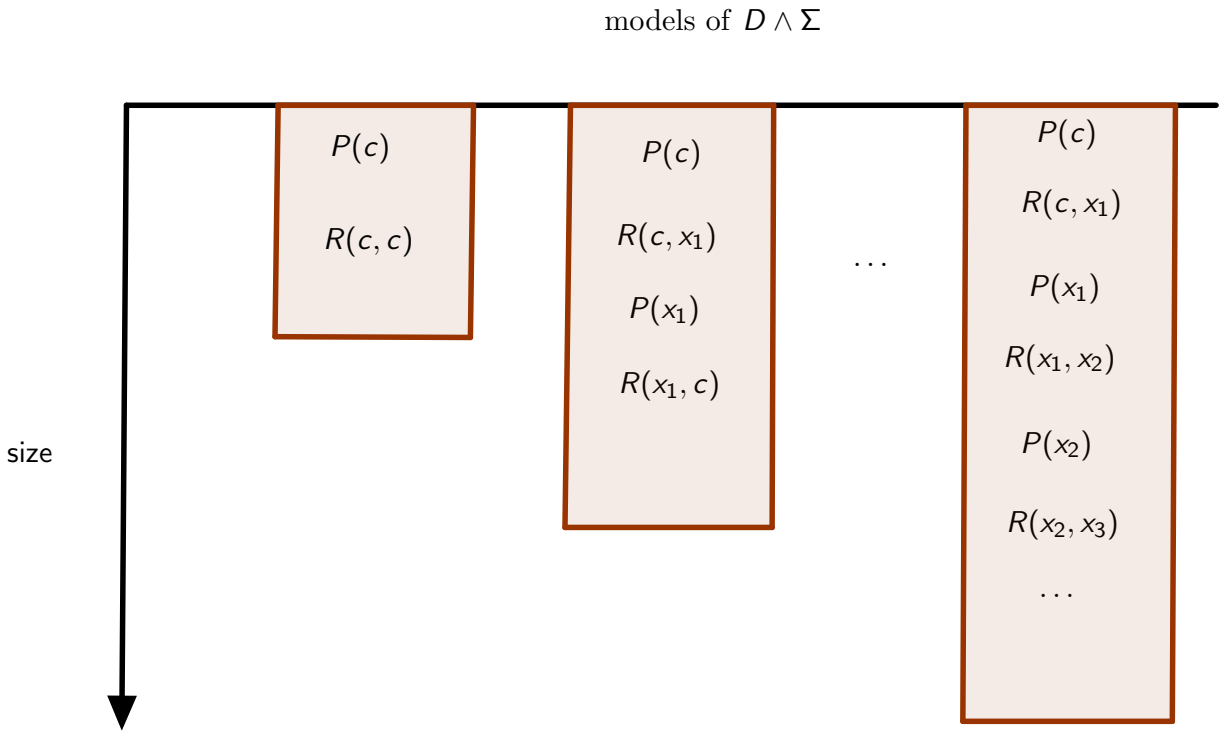
# Two Dimensions of Infinity

Consider $D = \{P(c)\}$ and
$\Sigma = \{\forall X(P(X) \rightarrow \exists Y(R(X, Y) \wedge P(Y)))\}$.



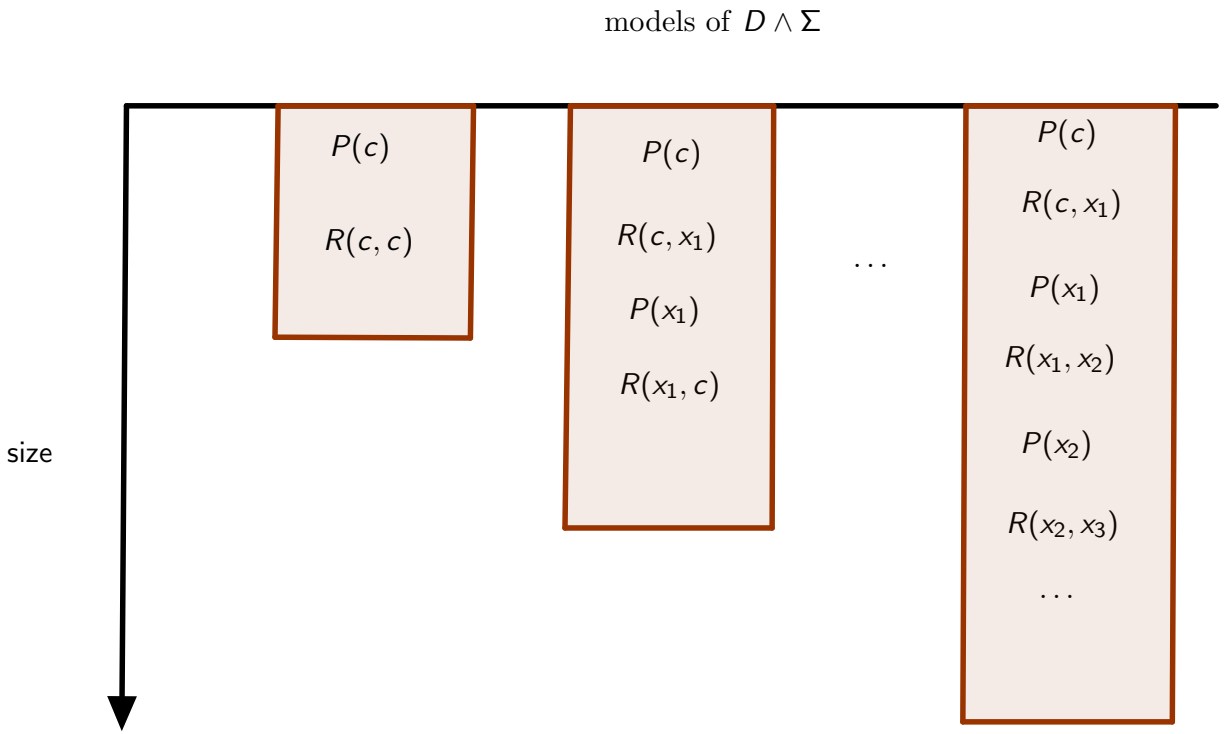
# Two Dimensions of Infinity

Consider $D = \{P(c)\}$ and
$\Sigma = \{\forall X(P(X) \rightarrow \exists Y(R(X, Y) \wedge P(Y)))\}$.



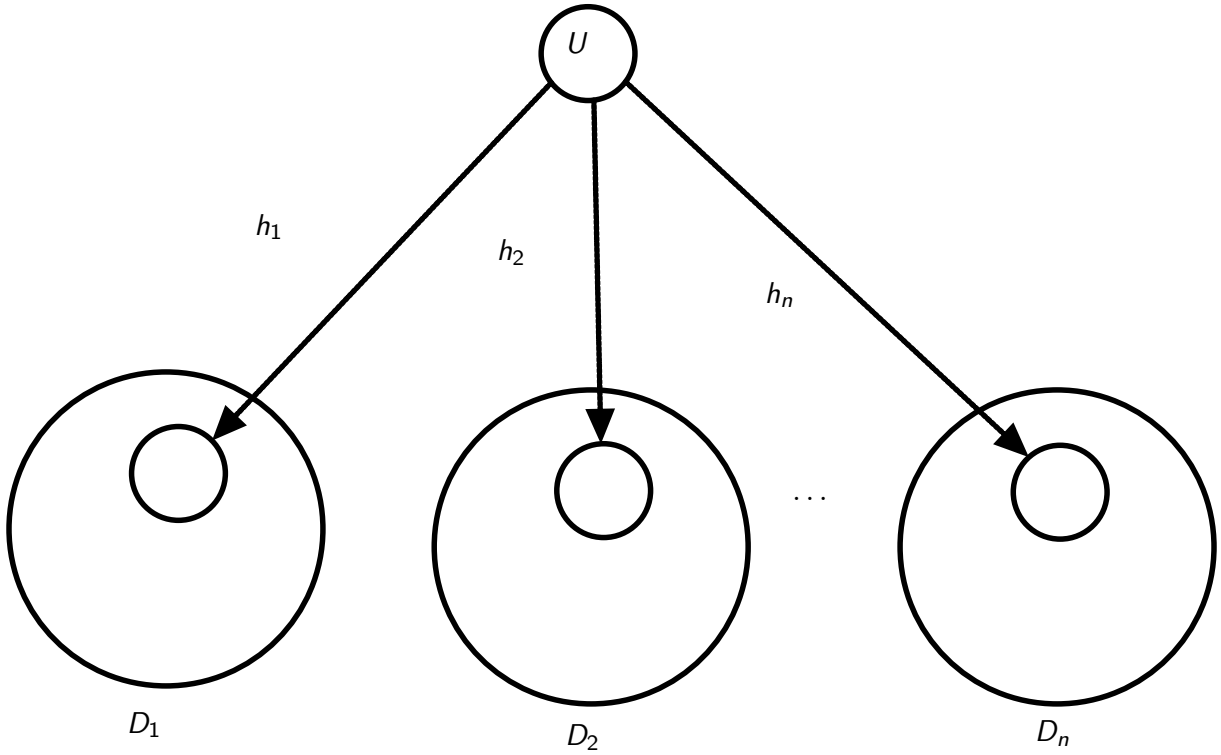
# Taming One Dimension of Infinity

Consider $D = \{P(c)\}$ and
$\Sigma = \{\forall X(P(X) \rightarrow \exists Y(R(X,Y) \wedge P(Y)))\}$.




**Key Idea**: focus on a representative, a model as general as possible.

# Universal Models (a.k.a. Canonical Models)




A database $U$ is a universal model of $D \wedge \Sigma$ if the following holds:

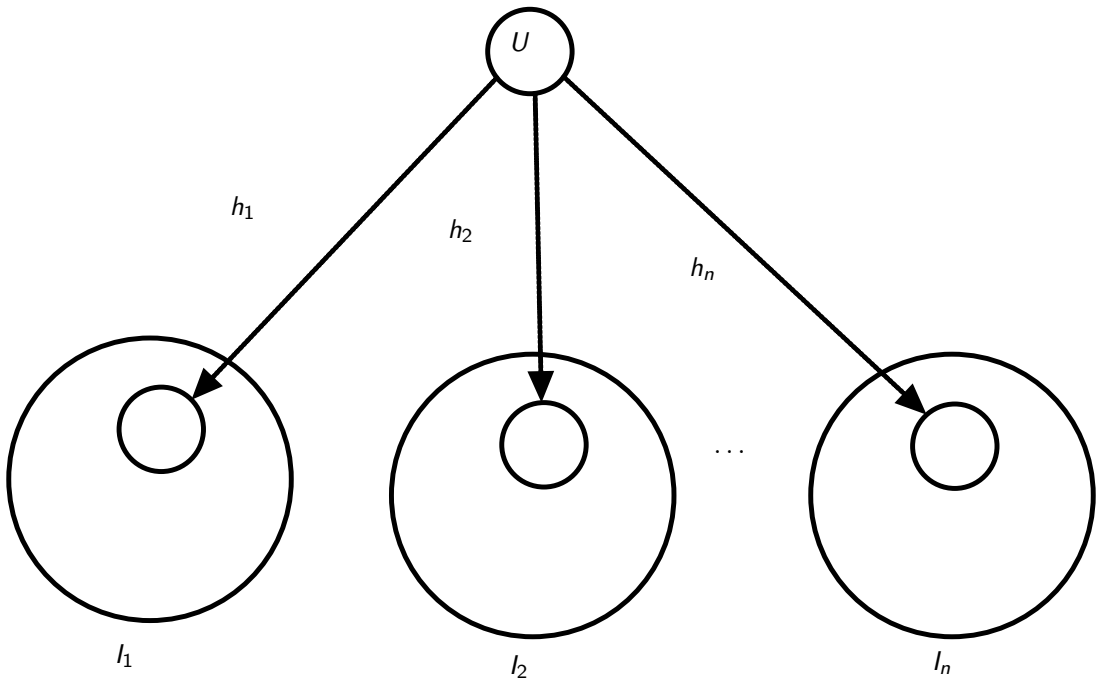
- $U$ is a model of $D \wedge \Sigma$;
- for any model $D'$ of $D \wedge \Sigma$, there exists a homomorphism $h_{D'}$ from $U$ to $D'$.

# Links between Query Answering and Universal Models

## Theorem

If $U$ is a universal model of $D \wedge \Sigma$, then, for any homomorphism closed query, it holds that:

$$D \wedge \Sigma \models q \text{ iff } U \models q$$

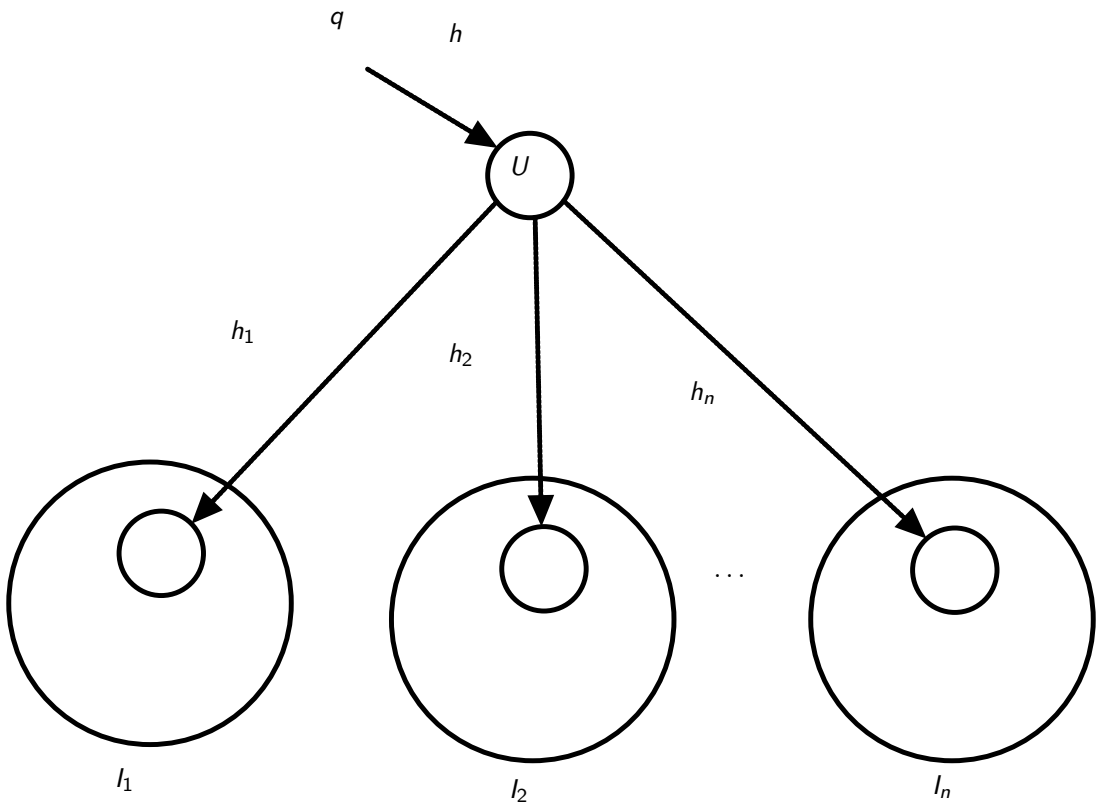


# Links between Query Answering and Universal Models

## Theorem

If $U$ is a universal model of $D \wedge \Sigma$, then, for any homomorphism closed query, it holds that:

$$D \wedge \Sigma \models q \text{ iff } U \models q$$



# Choice of the Ontology Language

From now on, we focus on *existential rules*.

# Choice of the Ontology Language

From now on, we focus on *existential rules*.

The reason for this choice it that it guarantees the existence of a universal model.
Exercise: show that this property does not hold for some ontology languages that we have seen in this course.

# The Chase Procedure: an Example

Let us see the *chase* on an example:

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

# The Chase Procedure: an Example

Let us see the *chase* on an example:

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\text{chase}(D, \Sigma) = D$$

# The Chase Procedure: an Example

Let us see the *chase* on an example:

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\text{chase}(D, \Sigma) = D \cup \{\text{hasParent}(\mathit{Alice}, x_1), \text{Person}(x_1)\}$$

# The Chase Procedure: an Example

Let us see the *chase* on an example:

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\begin{aligned}\text{chase}(D, \Sigma) = D \cup &\{\text{hasParent}(\mathit{Alice}, x_1), \text{Person}(x_1)\} \\ &\cup \{\text{hasParent}(x_1, x_2), \text{Person}(x_2)\}\end{aligned}$$

# The Chase Procedure: an Example

Let us see the *chase* on an example:

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\begin{aligned}\text{chase}(D, \Sigma) = D &\cup \{\text{hasParent}(\mathit{Alice}, x_1), \text{Person}(x_1)\} \\ &\cup \{\text{hasParent}(x_1, x_2), \text{Person}(x_2)\} \\ &\cup \{\text{hasParent}(x_2, x_3), \text{Person}(x_3)\}\end{aligned}$$

# The Chase Procedure: an Example

Let us see the *chase* on an example:

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\begin{aligned}\text{chase}(D, \Sigma) = D &\cup \{\text{hasParent}(\mathit{Alice}, x_1), \text{Person}(x_1)\} \\ &\cup \{\text{hasParent}(x_1, x_2), \text{Person}(x_2)\} \\ &\cup \{\text{hasParent}(x_2, x_3), \text{Person}(x_3)\} \\ &\dots\end{aligned}$$

In that case, an infinite database is generated.

# Rule Applicability

A rule $\sigma = \forall \mathbf{X} \forall \mathbf{Y}(\varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}(\psi(\mathbf{X}, \mathbf{Z})))$ is applicable to a database $D$ if:

1. there exists a homomorphism $h$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$;
2. there is no extension $h'$ of $h$ such that $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq D$.

## Example

- $D = \{R(a), P(a, b)\}$ and $\Sigma = \{\forall X(R(X) \rightarrow \exists Y\ P(X, Y))\}$
- $D = \{R(a), P(b, a)\}$ and $\Sigma = \{\forall X(R(X) \rightarrow \exists Y\ P(X, Y))\}$

# Rule Application

Let $\sigma$ be a rule applicable to $D$ through a homomorphism $\pi$ of its body. The result of the application of $\sigma$ to $D$ is defined as $D \cup \pi^s(H)$, where:

- $H$ is the head of $\sigma$
- $\pi^s(x) = \pi(x)$ if $x$ appears in the body of $\sigma$, and is a fresh null otherwise.

Example: $D = \{R(a), P(b, a)\}$ and $\Sigma = \{\forall X(R(X) \rightarrow \exists Y\ P(X, Y))\}$

$$\{R(a), P(b, a), P(a, \bot_1)\}$$

# Chase Sequence

A chase sequence of $D$ w.r.t. $\Sigma$ is a potentially infinite sequence $D_0, (\sigma_1, h_1), D_1, \ldots, D_n, \ldots$ such that:

- $\sigma_i$ is applicable to $D_{i-1}$ through $h_i$
- $D_i$ is the result of the application of $\sigma_i$ to $D_{i-1}$ through $h_i$
- for any $(\sigma, h)$ that is applicable to $D_j$ for some $j$, there exists $k > j$ such that $(\sigma, h)$ is not applicable to $D_k$

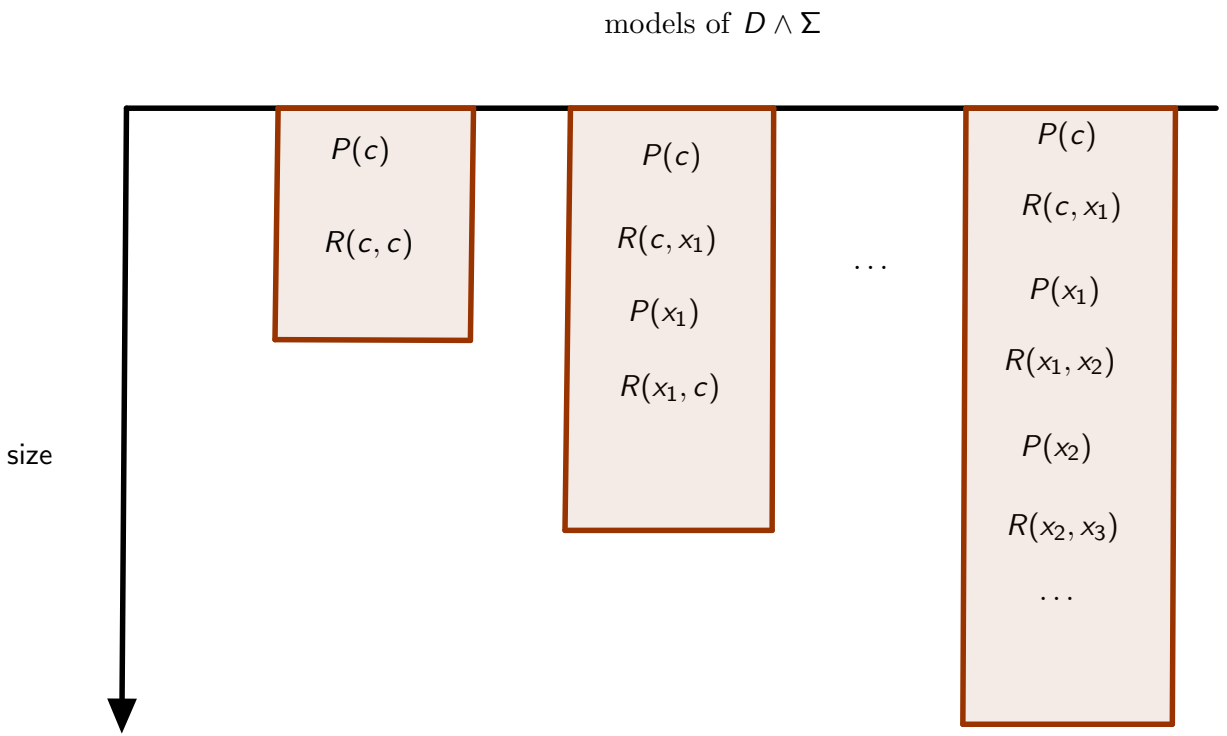
  This is called *fairness*.

The result of a chase sequence is either the last database in the sequence, or $\bigcup_{i=1}^{\infty} D_i$ if the sequence is infinite.

## Properties of the Chase

For any two chase sequences, their results are homomorphically equivalent, and are a univeral model of $D$ and $\Sigma$. We will denote the result by $chase(D, \Sigma)$.

# What about the Second Dimension of Infinity?




Intrisic difficulty: OBQA is undecidable.

# Gaining Decidability

When faced with an undecidable problem, one may want to *restrict the input* to regain decidability.

**By Restricting the database**

- the problem is already undecidable for singleton databases;
- not much to do in this direction!

**By Restricting the query language**

- the problem is already undecidable for atomic queries;
- not much to do in this direction!

**By Restricting the ontology language**

- we then aim at achieving a good trade-off between expressive power and complexity;
- lot of research has been done in this direction;
- any idea?

# Chase Termination

Let us revisit the chase example we have seen before.

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

# Chase Termination

Let us revisit the chase example we have seen before.

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\text{chase}(D, \Sigma) = D$$

# Chase Termination

Let us revisit the chase example we have seen before.

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\text{chase}(D, \Sigma) = D \cup \{\text{hasParent}(\mathit{Alice}, x_1), \text{Person}(x_1)\}$$

# Chase Termination

Let us revisit the chase example we have seen before.

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\begin{aligned}\text{chase}(D, \Sigma) = D \cup \{&\text{hasParent}(\mathit{Alice}, x_1), \text{Person}(x_1)\} \\ \cup \{&\text{hasParent}(x_1, x_2), \text{Person}(x_2)\}\end{aligned}$$

# Chase Termination

Let us revisit the chase example we have seen before.

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\begin{aligned}\text{chase}(D, \Sigma) = D &\cup \{\text{hasParent}(\mathit{Alice}, x_1), \text{Person}(x_1)\} \\ &\cup \{\text{hasParent}(x_1, x_2), \text{Person}(x_2)\} \\ &\cup \{\text{hasParent}(x_2, x_3), \text{Person}(x_3)\}\end{aligned}$$

# Chase Termination

Let us revisit the chase example we have seen before.

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\begin{aligned}\text{chase}(D, \Sigma) = D &\cup \{\text{hasParent}(\mathit{Alice}, x_1), \text{Person}(x_1)\} \\ &\cup \{\text{hasParent}(x_1, x_2), \text{Person}(x_2)\} \\ &\cup \{\text{hasParent}(x_2, x_3), \text{Person}(x_3)\} \\ &\dots\end{aligned}$$

# Chase Termination

Let us revisit the chase example we have seen before.

- $D = \{\text{Person}(\text{Alice})\}$
- $\Sigma = \{\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))\}$

$$\begin{aligned}\text{chase}(D, \Sigma) = D &\cup \{\text{hasParent}(\mathit{Alice}, x_1), \text{Person}(x_1)\} \\ &\cup \{\text{hasParent}(x_1, x_2), \text{Person}(x_2)\} \\ &\cup \{\text{hasParent}(x_2, x_3), \text{Person}(x_3)\} \\ &\dots\end{aligned}$$

Two ingredients of non-termination:

- existential quantification;
- recursive definitions.

# Syntactic Conditions to Ensure Chase Termination

Drop the existential quantification

- we obtain the class of *full* existential rules

Drop (or limit) the recursive definitions:

- plethora of acyclicity conditions defined
- see exercise sheets regarding this.

# Full Existential Rules

A full existential rule is an existential rule of the shape:

$$\forall \mathbf{X} \forall \mathbf{Y} (\varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \psi(\mathbf{X}))$$

We denote by FULL the class of full existential rules.
Belonging to FULL is:

- a local property: one can check each rule independently;
- and hence closed under union.

# Upper Bound on the Chase under FULL

Consider a database $D$ and a set $\Sigma \in$ FULL.

$$chase(D, \Sigma) \subseteq \{P(c_1, \ldots, c_n) \mid (c_1, \ldots, c_n) \in terms(D) \text{ and } P \in sch(\Sigma)\}$$

Hence, the size of $chase(D, \Sigma)$ is upper bounded by:

$$|sch(\Sigma)| \times |terms(D)|^{maxarity(sch(\Sigma))}$$

# Data Complexity of BCQ Answering under FULL

**Theorem**

Boolean Conjuctive Query (BCQ) Answering under FULL is in PTIME in data complexity.

Proof idea:

1. compute the chase of $D$ and $\Sigma$;
2. check whether there is a homomorphism from $q$ to $chase(D, \Sigma)$.

# Data Complexity of BCQ Answering under FULL

**Theorem**

Boolean Conjuctive Query (BCQ) Answering under FULL is in PTIME in data complexity.

Proof idea:

1. compute the chase of $D$ and $\Sigma$;
2. check whether there is a homomorphism from $q$ to $chase(D, \Sigma)$.

Can we do any better?

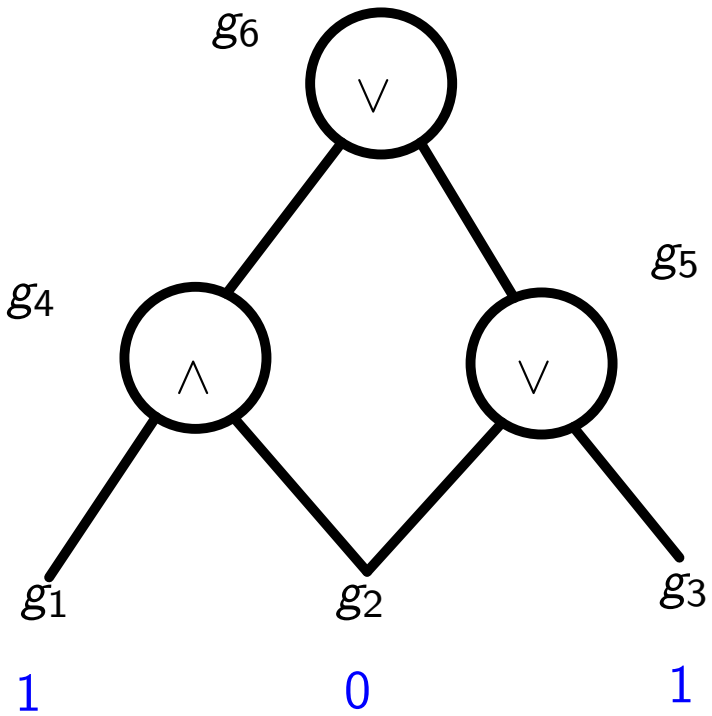
# Data Complexity of BCQ Answering under FULL

**Theorem**

Boolean Conjuctive Query (BCQ) Answering under FULL is PTIME-hard in data complexity.

Proof idea: by a LOGSPACE reduction from Monotone Circuit Value problem.

# Data Complexity of BCQ Answering under FULL

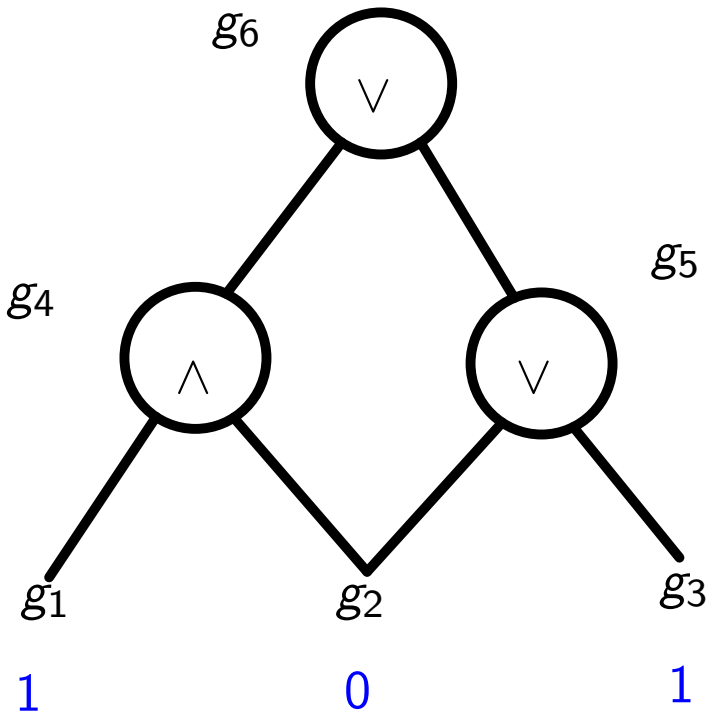


# Data Complexity of BCQ Answering under FULL

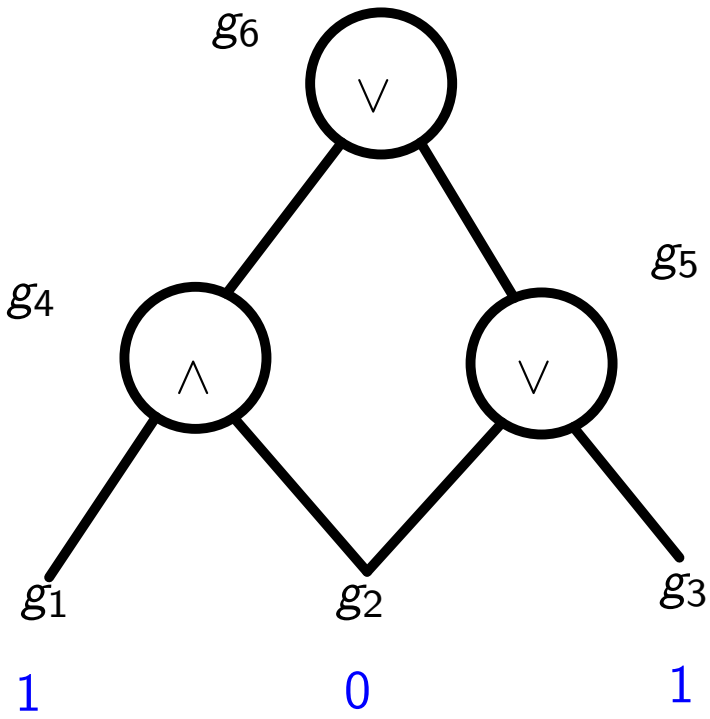
**Encoding the circuit as:**

$$T(g_1) \quad T(g_3)$$

$$AND(g_4, g_1, g_2), OR(g_5, g_2, g_3), OR(g_6, g_4, g_5)$$



# Data Complexity of BCQ Answering under FULL




**Encoding the circuit as:**

$$T(g_1) \quad T(g_3)$$

$$AND(g_4, g_1, g_2), OR(g_5, g_2, g_3), OR(g_6, g_4, g_5)$$

**Evaluation of the circuit via a fixed set:**

$$\forall X \forall Y \forall Z (T(X) \wedge OR(Z, X, Y) \rightarrow T(Z))$$

$$\forall X \forall Y \forall Z (T(Y) \wedge OR(Z, X, Y) \rightarrow T(Z))$$

$$\forall X \forall Y \forall Z (T(X) \wedge T(Y) \wedge AND(Z, X, Y) \rightarrow T(Z))$$

# Combined Complexity of BCQ Answering under FULL

We can show that the problem is ExpTime-complete by simulating an ExpTime Turing machine.
Adaptation of what we have already seen:

- not possible anymore to create new tapes using existential
- hence we use two counters:
  - one for the cell address
  - one as a time stamp

# Alternative Approaches

- stop the chase at some arbitrary step, and work with the obtained results:
  - not very satisfying...
- stop the chase for some reason, and get the complete set of results:
  - what could be a criterion for stopping the chase?
- avoid any chase computation, and rely on a different paradigm:
  - this is the approach that we are going to see during the next course.

# The DL-Lite Family and OWL 2 QL

- OWL 2 QL is the OWL 2 profile designed for efficient query answering
- Target large datasets: CQ answering is in AC0 in data complexity (below $\textsc{LogSpace}$)
- Based on the DL-Lite$_{\mathcal{R}}$ language of the DL-Lite family

# The DL-Lite Family and OWL 2 QL

DL-Lite$_{\text{core}}$ : concept inclusions of the form $B \sqsubseteq C$ where

$$C := B \mid \neg B, \quad B := A \mid \exists S, \quad S := R \mid R^{-}$$

with $A$ an atomic concept and $R$ an atomic role

# The DL-Lite Family and OWL 2 QL

DL-Lite$_{\text{core}}$ : concept inclusions of the form $B \sqsubseteq C$ where

$$C := B \mid \neg B, \quad B := A \mid \exists S, \quad S := R \mid R^-$$

with $A$ an atomic concept and $R$ an atomic role

Several extensions of DL-Lite$_{\text{core}}$, among which:

- DL-Lite$_{\mathcal{R}}$ = DL-Lite$_{\text{core}}$ + role inclusions $S \sqsubseteq Q$ with $Q := S \mid \neg S$
- DL-Lite$_{\mathcal{F}}$ = DL-Lite$_{\text{core}}$ + functionality axioms $(\mathit{func}\ S)$

# Linear Rules

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

- Linear existential rules have only one atom in the body
- Generalize DL-Lite$_{\mathcal{R}}$ restricted to positive inclusions (of the form $B_1 \sqsubseteq B_2$ or $S_1 \sqsubseteq S_2$)

# Relationships between DL-Lite and Linear Rules

| DL-Lite$_\mathcal{R}$ | Linear rule |
|---|---|
| $A_1 \sqsubseteq A_2$ | $A_1(x) \rightarrow A_2(x)$ |
| $R_1 \sqsubseteq R_2$ | $R_1(x, y) \rightarrow R_2(x, y)$ |
| $R_1 \sqsubseteq R_2^-$ | $R_1(x, y) \rightarrow R_2(y, x)$ |
| $A \sqsubseteq \exists R$ | $A(x) \rightarrow R(x, y)$ |
| $\exists R \sqsubseteq A$ | $R(x, y) \rightarrow A(x)$ |
| ... | |

Negative inclusions $B_1 \sqsubseteq \neg B_2$ or $S_1 \sqsubseteq \neg S_2$ are not expressible by linear rules: need to add negative constraints of the form

$$\forall \vec{X} \forall \vec{Y} (\phi(\vec{X}, \vec{Y}) \rightarrow \bot)$$

Functionality axioms present in DL-Lite$_\mathcal{F}$ are not expressible either: need to add equality rules

$$\forall \vec{X} \forall X_1, X_2 (\phi(\vec{X}, X_1, X_2) \rightarrow X_1 = X_2)$$

# Linear Rules

Linear rule:

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

# Linear Rules

Linear rule:
$\forall X(\mathrm{Person}(X) \rightarrow \exists Y(\mathrm{hasParent}(X, Y) \wedge \mathrm{Person}(Y)))$

In DL syntax: $\mathsf{Person} \sqsubseteq \exists \mathsf{hasParent}.\mathsf{Person}$

Can be expressed in DL-Lite:

$$\mathsf{Person} \sqsubseteq \exists \mathsf{hasParentPerson}$$
$$\exists \mathsf{hasParentPerson}^{-} \sqsubseteq \mathsf{Person}$$
$$\mathsf{hasParentPerson} \sqsubseteq \mathsf{hasParent}$$

# Linear Rules

Linear rule:
$\forall X(\text{Person}(X) \rightarrow \exists Y(\text{hasParent}(X, Y) \wedge \text{Person}(Y)))$

In DL syntax: Person $\sqsubseteq \exists$hasParent.Person

Can be expressed in DL-Lite:

$$\begin{aligned} \text{Person} &\sqsubseteq \exists \text{hasParentPerson} \\ \exists \text{hasParentPerson}^{-} &\sqsubseteq \text{Person} \\ \text{hasParentPerson} &\sqsubseteq \text{hasParent} \end{aligned}$$

We have seen that the chase may not terminate with this rule, however, CQ answering can be done very efficiently in DL-Lite$_{\mathcal{R}}$!

# Query Rewriting

Instead of computing $chase(D, \Sigma)$ and evaluate a query $q$ on it, we compute a query $q'$ such that:

$$\langle \Sigma, D \rangle \models q \Leftrightarrow D \models q'$$

- the dataset (ABox, set of facts) is stored as a traditional database
- the input query is rewritten to integrate the relevant information from the ontology
- the new query is evaluated over the database
- allow to exploit the efficiency of relational database systems

# Query Rewriting

A query $q_\Sigma$ is called a sound and complete rewriting of a CQ $q$ w.r.t. an ontology $\Sigma$ if and only if for every database $D$, the certain answers of $q$ over $\langle \Sigma, D \rangle$ are the same as the answers of $q_\Sigma$ over $D$.

# Query Rewriting

A query $q_\Sigma$ is called a sound and complete rewriting of a CQ $q$ w.r.t. an ontology $\Sigma$ if and only if for every database $D$, the certain answers of $q$ over $\langle \Sigma, D \rangle$ are the same as the answers of $q_\Sigma$ over $D$.

If for every $\Sigma$ expressed in an ontology language $L_\Sigma$ and CQ $q$, there exists a sound and complete rewriting which belongs to a query language $L_Q$, we say that CQ answering under $L_\Sigma$ is $L_Q$-rewritable.

# Query Rewriting

A query $q_\Sigma$ is called a sound and complete rewriting of a CQ $q$ w.r.t. an ontology $\Sigma$ if and only if for every database $D$, the certain answers of $q$ over $\langle \Sigma, D \rangle$ are the same as the answers of $q_\Sigma$ over $D$.

If for every $\Sigma$ expressed in an ontology language $L_\Sigma$ and CQ $q$, there exists a sound and complete rewriting which belongs to a query language $L_Q$, we say that CQ answering under $L_\Sigma$ is $L_Q$-rewritable.

CQ answering under linear existential rules is UCQ-rewritable.

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild},$$
$$\text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$
$$\mathcal{A} = \{ \ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \ \}$$
$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild},$$
$$\text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \ \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) =$$
$$(\exists y \ \text{Person}(x) \wedge \text{hasChild}(x, y))$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild},$$
$$\text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) =$$
$$(\exists y \ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y \ \text{Parent}(x) \wedge \text{hasChild}(x, y))$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild}, \qquad \text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) = (\exists y \ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y \ \text{Parent}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y \ \text{Mother}(x) \wedge \text{hasChild}(x, y))$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild}, \quad \text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) = (\exists y \ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y \ \text{Parent}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y \ \text{Mother}(x) \wedge \text{hasChild}(x, y)) \vee (\text{Person}(x) \wedge \text{Parent}(x))$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild}, \quad \text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) = (\exists y\ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y\ \text{Parent}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y\ \text{Mother}(x) \wedge \text{hasChild}(x, y)) \vee (\text{Person}(x) \wedge \text{Parent}(x)) \vee \text{Parent}(x)$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild}, \quad \text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) = (\exists y\ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y\ \text{Parent}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y\ \text{Mother}(x) \wedge \text{hasChild}(x, y)) \vee (\text{Person}(x) \wedge \text{Parent}(x)) \vee \text{Parent}(x) \vee (\text{Mother}(x) \wedge \text{Parent}(x))$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild},$$
$$\text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) =$$
$$(\exists y\ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y\ \text{Parent}(x) \wedge \text{hasChild}(x, y))$$
$$\vee (\exists y\ \text{Mother}(x) \wedge \text{hasChild}(x, y)) \vee (\text{Person}(x) \wedge \text{Parent}(x))$$
$$\vee \text{Parent}(x) \vee (\text{Mother}(x) \wedge \text{Parent}(x)) \vee \text{Mother}(x)$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild}, \qquad \text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) =$$

$$(\exists y\ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y\ \text{Parent}(x) \wedge \text{hasChild}(x, y))$$
$$\vee (\exists y\ \text{Mother}(x) \wedge \text{hasChild}(x, y)) \vee (\text{Person}(x) \wedge \text{Parent}(x))$$
$$\vee \text{Parent}(x) \vee (\text{Mother}(x) \wedge \text{Parent}(x)) \vee \text{Mother}(x)$$
$$\vee (\exists yz\ \text{isMarriedTo}(x, z) \wedge \text{hasChild}(x, y))$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild}, \quad \text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) =$$

$$(\exists y\ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y\ \text{Parent}(x) \wedge \text{hasChild}(x, y))$$
$$\vee (\exists y\ \text{Mother}(x) \wedge \text{hasChild}(x, y)) \vee (\text{Person}(x) \wedge \text{Parent}(x))$$
$$\vee \text{Parent}(x) \vee (\text{Mother}(x) \wedge \text{Parent}(x)) \vee \text{Mother}(x)$$
$$\vee (\exists yz\ \text{isMarriedTo}(x, z) \wedge \text{hasChild}(x, y))$$
$$\vee (\exists z\ \text{isMarriedTo}(x, z) \wedge \text{Parent}(x))$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild},$$
$$\text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) =$$
$$(\exists y\ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y\ \text{Parent}(x) \wedge \text{hasChild}(x, y))$$
$$\vee (\exists y\ \text{Mother}(x) \wedge \text{hasChild}(x, y)) \vee (\text{Person}(x) \wedge \text{Parent}(x))$$
$$\vee \text{Parent}(x) \vee (\text{Mother}(x) \wedge \text{Parent}(x)) \vee \text{Mother}(x)$$
$$\vee (\exists yz\ \text{isMarriedTo}(x, z) \wedge \text{hasChild}(x, y))$$
$$\vee (\exists z\ \text{isMarriedTo}(x, z) \wedge \text{Parent}(x))$$
$$\vee (\exists z\ \text{isMarriedTo}(x, z) \wedge \text{Mother}(x))$$

# Query Rewriting

## Example

$$\mathcal{T} = \{ \quad \text{Mother} \sqsubseteq \text{Parent}, \qquad \text{Parent} \sqsubseteq \exists \text{hasChild},$$
$$\text{Parent} \sqsubseteq \text{Person}, \qquad \exists \text{isMarriedTo} \sqsubseteq \text{Person} \quad \}$$

$$\mathcal{A} = \{ \ \text{Mother}(\mathit{mary}), \text{hasChild}(\mathit{alice}, \mathit{john}), \text{isMarriedTo}(\mathit{alice}, \mathit{bob}) \ \}$$

$$q(x) = \exists y \text{Person}(x) \wedge \text{hasChild}(x, y)$$

$$q'(x) =$$
$$(\exists y \ \text{Person}(x) \wedge \text{hasChild}(x, y)) \vee (\exists y \ \text{Parent}(x) \wedge \text{hasChild}(x, y))$$
$$\vee (\exists y \ \text{Mother}(x) \wedge \text{hasChild}(x, y)) \vee (\text{Person}(x) \wedge \text{Parent}(x))$$
$$\vee \text{Parent}(x) \vee (\text{Mother}(x) \wedge \text{Parent}(x)) \vee \text{Mother}(x)$$
$$\vee (\exists yz \ \text{isMarriedTo}(x, z) \wedge \text{hasChild}(x, y))$$
$$\vee (\exists z \ \text{isMarriedTo}(x, z) \wedge \text{Parent}(x))$$
$$\vee (\exists z \ \text{isMarriedTo}(x, z) \wedge \text{Mother}(x))$$

# Simplifying Assumption

For now, we consider normalized existential rules, where only one atom appears in the head.

Every existential rule can be transformed into a set of normalized rules equivalent w.r.t. query answering.

$$\phi(\vec{X}, \vec{Y}) \rightarrow P_1(\vec{X}, \vec{Z}) \wedge \cdots \wedge P_n(\vec{X}, \vec{Z})$$

$$\Downarrow$$

$$\phi(\vec{X}, \vec{Y}) \rightarrow \mathrm{Aux}(\vec{X}, \vec{Z})$$

$$\mathrm{Aux}(\vec{X}, \vec{Z}) \rightarrow P_1(\vec{X}, \vec{Z})$$

$$\vdots$$

$$\mathrm{Aux}(\vec{X}, \vec{Z}) \rightarrow P_n(\vec{X}, \vec{Z})$$

# Rewriting Step

## Datalog rewriting (Datalog = FULL = no existential)

$$\Sigma = \{\text{hasCollaborator}(X, Y, Z) \rightarrow \text{Collaborator}(X)\}$$
$$q = \text{Collaborator}(a) \wedge \text{worksFor}(a, b)$$

Unify head rule and query atom $\mu = \{X \mapsto a\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \text{Collaborator}(a) \wedge \text{worksFor}(a, b)$$
$$\vee$$
$$\exists YZ\ \text{hasCollaborator}(a, Y, Z) \wedge \text{worksFor}(a, b)$$

# Rewriting Step

## Existential in the head

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists UV \ \text{hasCollaborator}(U, b, V)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto U\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \exists UV \ \text{hasCollaborator}(U, b, V)$$
$$\vee$$
$$\exists V \ \text{Project}(V) \wedge \text{inArea}(V, b)$$

# Rewriting Step

## Existential in the head : unsound rewriting

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists V \ \text{hasCollaborator}(a, b, V)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto a\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \exists V \ \text{hasCollaborator}(a, b, V)$$
$$\vee$$
$$\exists V \ \text{Project}(V) \wedge \text{inArea}(V, b)$$

# Rewriting Step

## Existential in the head : unsound rewriting

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists V \ \text{hasCollaborator}(a, b, V)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto a\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \exists V \ \text{hasCollaborator}(a, b, V)$$
$$\vee$$
$$\exists V \ \text{Project}(V) \wedge \text{inArea}(V, b)$$

Consider $D = \{\text{Project}(c), \text{inArea}(c, b)\}$

$D \models q_\Sigma$ but $\langle \Sigma, D \rangle \not\models q$ : $q_\Sigma$ is not sound !

# Rewriting Step

## Existential in the head : unsound rewriting

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists V \ \text{hasCollaborator}(a, b, V)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto a\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \exists V \ \text{hasCollaborator}(a, b, V)$$
$$\vee$$
$$\exists V \ \text{Project}(V) \wedge \text{inArea}(V, b)$$

Consider $D = \{\text{Project}(c), \text{inArea}(c, b)\}$

$D \models q_\Sigma$ but $\langle \Sigma, D \rangle \not\models q$ : $q_\Sigma$ is not sound !

Problem: constant $a$ has been unified with an existential variable

# Rewriting Step

## Existential in the head : unsound rewriting

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists V \ \text{hasCollaborator}(V, b, V)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto V\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \exists V \ \text{hasCollaborator}(V, b, V)$$
$$\vee$$
$$\exists V \ \text{Project}(V) \wedge \text{inArea}(V, b)$$

# Rewriting Step

## Existential in the head : unsound rewriting

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists V \ \text{hasCollaborator}(V, b, V)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto V\}$
Replace query atom by $\mu(body(rule))$

$$\begin{aligned} q_\Sigma = & \exists V \ \text{hasCollaborator}(V, b, V) \\ & \vee \\ & \exists V \ \text{Project}(V) \wedge \text{inArea}(V, b) \end{aligned}$$

Consider $D = \{\text{Project}(c), \text{inArea}(c, b)\}$

$D \models q_\Sigma$ but $\langle \Sigma, D \rangle \not\models q$ : $q_\Sigma$ is not sound !

# Rewriting Step

## Existential in the head : unsound rewriting

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists V \ \text{hasCollaborator}(V, b, V)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto V\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \exists V \ \text{hasCollaborator}(V, b, V)$$
$$\vee$$
$$\exists V \ \text{Project}(V) \wedge \text{inArea}(V, b)$$

Consider $D = \{\text{Project}(c), \text{inArea}(c, b)\}$

$D \models q_\Sigma$ but $\langle \Sigma, D \rangle \not\models q$ : $q_\Sigma$ is not sound !

Problem: existential variable $Z$ has been unified with $X$

# Rewriting Step

## Existential in the head : unsound rewriting

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists U V \ \text{hasCollaborator}(U, b, V) \wedge \text{Manager}(U)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto U\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \exists U V \ \text{hasCollaborator}(U, b, V) \wedge \text{Manager}(U)$$
$$\vee$$
$$\exists U V \ \text{Project}(V) \wedge \text{inArea}(V, b) \wedge \text{Manager}(U)$$

# Rewriting Step

## Existential in the head : unsound rewriting

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists U V \ \text{hasCollaborator}(U, b, V) \wedge \text{Manager}(U)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto U\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \exists U V \ \text{hasCollaborator}(U, b, V) \wedge \text{Manager}(U)$$
$$\vee$$
$$\exists U V \ \text{Project}(V) \wedge \text{inArea}(V, b) \wedge \text{Manager}(U)$$

Consider $D = \{\text{Project}(c), \text{inArea}(c, b), \text{Manager}(Alice)\}$

$D \models q_\Sigma$ but $\langle \Sigma, D \rangle \not\models q$ : $q_\Sigma$ is not sound !

# Rewriting Step

## Existential in the head : unsound rewriting

$$\Sigma = \{\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists U V \ \text{hasCollaborator}(U, b, V) \wedge \text{Manager}(U)$$

Unify head rule and query atom $\mu = \{X \mapsto V, Y \mapsto b, Z \mapsto U\}$
Replace query atom by $\mu(body(rule))$

$$q_\Sigma = \exists U V \ \text{hasCollaborator}(U, b, V) \wedge \text{Manager}(U)$$
$$\vee$$
$$\exists U V \ \text{Project}(V) \wedge \text{inArea}(V, b) \wedge \text{Manager}(U)$$

Consider $D = \{\text{Project}(c), \text{inArea}(c, b), \text{Manager}(Alice)\}$

$D \models q_\Sigma$ but $\langle \Sigma, D \rangle \not\models q$ : $q_\Sigma$ is not sound !

Problem: an atom containing the a variable unified with the existential variable is not unified.

# Applicability conditions

What if we add conditions to apply a rewriting step to a query atom ?

1. only universal variables of the rule head can be unified with a constant
2. only universal variables of the rule head can be unified with a shared variable of the query
3. an existential variable of the rule head cannot be unified with another variable of the rule head

# Incomplete rewriting

$$\Sigma = \{\text{hasCollaborator}(X, Y, Z) \rightarrow \text{Collaborator}(X),$$
$$\text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}$$
$$q = \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U)$$

$$q_\Sigma = \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U)$$
$$\vee$$
$$\exists UVWYZ\ \text{hasCollaborator}(U, V, W) \wedge \text{hasCollaborator}(U, Y, Z)$$

# Incomplete rewriting

$$\begin{aligned}\Sigma =& \{\text{hasCollaborator}(X, Y, Z) \rightarrow \text{Collaborator}(X),\\ & \quad \text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\}\\ q =& \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U)\end{aligned}$$

$$\begin{aligned}q_\Sigma =& \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U)\\ & \vee\\ & \exists UVWYZ\ \text{hasCollaborator}(U, V, W) \wedge \text{hasCollaborator}(U, Y, Z)\end{aligned}$$

Consider $D = \{\text{Project}(a), \text{inArea}(a, b)\}$

$\langle \Sigma, D \rangle \models q$ but $D \not\models q_\Sigma$: $q_\Sigma$ is not complete !

Applicability conditions may destroy completeness !

# Minimization Step

$$\begin{aligned}\Sigma =& \{\text{hasCollaborator}(X, Y, Z) \rightarrow \text{Collaborator}(X) \\ & \text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\} \\ q =& \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U)\end{aligned}$$

$$\begin{aligned}q_\Sigma =& \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U) \\ & \vee \\ & \exists UVWYZ\ \text{hasCollaborator}(U, V, W) \wedge \text{hasCollaborator}(U, Y, Z)\end{aligned}$$

# Minimization Step

$$\begin{aligned}\Sigma =& \{\text{hasCollaborator}(X, Y, Z) \rightarrow \text{Collaborator}(X) \\ & \text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\} \\ q =& \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U)\end{aligned}$$

$$\begin{aligned}q_\Sigma =& \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U) \\ & \vee \\ & \exists UVWYZ\ \text{hasCollaborator}(U, V, W) \wedge \text{hasCollaborator}(U, Y, Z) \\ & \vee \\ & \exists UVW\ \text{hasCollaborator}(U, V, W) \text{ - by minimization}\end{aligned}$$

# Minimization Step

$$\begin{aligned}\Sigma =& \{\text{hasCollaborator}(X, Y, Z) \rightarrow \text{Collaborator}(X) \\ & \text{Project}(X) \wedge \text{inArea}(X, Y) \rightarrow \text{hasCollaborator}(Z, Y, X)\} \\ q =& \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U)\end{aligned}$$

$$\begin{aligned}q_\Sigma =& \exists UVW\ \text{hasCollaborator}(U, V, W) \wedge \text{Collaborator}(U) \\ & \vee \\ & \exists UVWYZ\ \text{hasCollaborator}(U, V, W) \wedge \text{hasCollaborator}(U, Y, Z) \\ & \vee \\ & \exists UVW\ \text{hasCollaborator}(U, V, W) \text{ - by minimization} \\ & \vee \\ & \exists VW\ \text{Project}(W) \wedge \text{inArea}(W, V)\end{aligned}$$

# Rewriting Algorithm

The standard algorithm for computing UCQ-rewritings performs an exhaustive application of the following two steps:

1. Rewriting
2. Minimization

# Piece Based Unification

A piece unifier of a query $q$ with an existential rule $\sigma$ is a triple $\mu = (q', H', P_u)$ such that:

- $q'$ is a non empty subset of $q$
- $H'$ is a subset of the head $H$ of $\sigma$
- $P_u$ is partition on the terms of $H'$ and $q'$ s.t.:
  - no two constants belong to the same class
  - if a class of $P_u$ contains an existential variable of $H'$, then the other terms are non-separating variables of $q'$ (*i.e.*, variables of $q'$ that do not appear in $q \setminus q'$)
    - note: no constants, no other existential variable of $H'$
  - $u(H') = u(q')$, where $u$ is a substitution being the identity on constants and assigning to two terms the same image if and only if their are in the same class in $P_u$.

# Piece Based Unification

Rewriting step: given a piece unifier $\mu = (q', H', P_u)$, $rewrite(q, \sigma, \mu)$ is the query obtained by replacing $q'$ by $u(body(\sigma))$ in $q$

# Piece Based Unification

Rewriting step: given a piece unifier $\mu = (q', H', P_u)$, $rewrite(q, \sigma, \mu)$ is the query obtained by replacing $q'$ by $u(body(\sigma))$ in $q$

Using piece based unification instead of atom unification

- replace minimization step+rewriting step by a single rewriting step that rewrites several axioms together
  - integrate the applicability conditions : sound rewriting
  - may rewrite several axioms that unify together: complete rewriting
- allow us to drop the simplifying assumption that rules contain a single atom in the head

# Piece Based Unification

## Theorem

$\langle \Sigma, D \rangle \models q$ iff there is a sequence $q_0 = q, q_1, \ldots, q_k$ such that

- for all $i$ with $0 \leq i < k$, there is $\sigma_i \in \Sigma$ and a piece unifier $\mu_i$ of $q_i$ with $\sigma_i$ such that $q_{i+1} = \mathit{rewrite}(q_i, \sigma_i, \mu_i)$
- $D \models q_k$

# Find the Piece Unifiers!

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

$$\begin{aligned}
q_1 &= P(a, a, b, c) \\
q_2 &= \exists VW \ P(a, b, V, W) \\
q_3 &= \exists UVW \ P(U, U, V, W) \\
q_4 &= \exists UVW \ P(U, V, W, W) \\
q_5 &= \exists UVW \ P(U, V, V, W) \\
q_6 &= \exists UVWH \ P(a, b, U, W) \wedge P(a, b, V, H) \\
q_7 &= \exists UVWH \ P(a, b, U, H) \wedge P(W, b, V, H) \\
q_8 &= \exists UVWHK \ P(U, V, W, H) \wedge P(U, U, V, K)
\end{aligned}$$

# A Naive Rewriting Algorithm

**Algorithm 1** Naive rewriting algorithm

$q_\Sigma = \emptyset$, $Rew = \{q\}$
**while** $Rew \neq q_\Sigma$ **do**
    $q_\Sigma = Rew$
    **for** $q_i \in Rew$ **do**
        **for** $\sigma \in \Sigma$ **do**
            **for** $\mu$ piece unifier of $q_i$ with $\sigma$ **do**
                $q_{rew} = rewrite(q_i, \sigma, \mu)$
                **if** $q_{rew} \notin Rew$ modulo variables renaming **then**
                    $Rew = Rew \cup \{q_{rew}\}$
Output $q_\Sigma$

# A Naive Rewriting Algorithm

If the naive rewriting algorithm terminates,
then $\langle \Sigma, D \rangle \models q$ iff $D \models q_\Sigma$

- Soundness : every $q' \in q_\Sigma$ is obtained by a sequence of piece based rewriting steps
- Completeness: if the algorithm terminates, then $q_\Sigma$ contains all queries that can be obtained by a sequence of piece based rewriting steps

# Proof of Termination for Linear Rules

- The size of $q_\Sigma$ grows at each iteration
- $q_\Sigma$ does not contain any two queries that are equal modulo variables renaming
- Since $|body(\sigma)| = 1$, then $|rewrite(q_i, \sigma, \mu)| \leq |q_i|$
- So $|q_\Sigma|$ is bounded by $(|sch(\Sigma)| \times maxterms^{maxarity})^{|q|}$ where $maxterms = maxarity \times |q|$

# Improving the Algorithm

- If $q_i$ has been rewritten, it is not useful to consider it again
- If there is an homomorphism from $q_1$ to $q_2$ and $q_\Sigma$ is a sound and complete rewriting of $q$, then $q_\Sigma \setminus \{q_2\}$ is also a sound and complete rewriting of $q$
- If there is an homomorphism from $q_1$ to $q_2$, then for every rewriting $q_2'$ of $q_2$, there exists a rewriting $q_1'$ of $q_1$ such that there is an homomorphism from $q_1'$ to $q_2'$

# Rewrite

$$\Sigma = \{\text{hasChild}(X, Y) \rightarrow \text{hasParent}(Y, X),$$
$$\text{hasChild}(X, Y) \rightarrow \text{Parent}(X),$$
$$\text{Spouse}(X) \rightarrow \text{isMarriedTo}(X, Y),$$
$$\text{isMarriedTo}(X, Y) \rightarrow \text{Spouse}(X),$$
$$\text{sisterOf}(X, Y) \rightarrow \text{siblingOf}(X, Y),$$
$$\text{siblingOf}(X, Y) \rightarrow \text{siblingOf}(Y, X),$$
$$\text{hasSisterInLaw}(X, Z) \rightarrow \text{isMarriedTo}(X, Y) \wedge \text{sisterOf}(Y, Z),$$
$$\text{hasParentInLaw}(X, Z) \rightarrow \text{isMarriedTo}(X, Y) \wedge \text{hasParent}(Y, Z)\}$$

$$q_1(x) = \exists yz \ \text{isMarriedTo}(x, y) \wedge \text{siblingOf}(y, z)$$

$$q_2(x) = \exists y \ \text{Parent}(x) \wedge \text{hasParent}(y, x) \wedge \text{Spouse}(y)$$

# Conjunctive Query Answering in $\mathcal{EL}$

- In $\mathcal{EL}$, the chase does not terminate
  - $\text{Person} \sqsubseteq \exists \text{hasParent}.\text{Person}$

# Conjunctive Query Answering in $\mathcal{EL}$

- In $\mathcal{EL}$, the chase does not terminate
  - $\text{Person} \sqsubseteq \exists \text{hasParent}.\text{Person}$
- The rewriting algorithm does not terminate either !
  - $\exists \text{hasParent}.\text{Person} \sqsubseteq \text{Person}$
- $\mathcal{EL}$ is not UCQ-rewritable
  - $q = \text{Person}(a)$
  - $\mathcal{T} = \{\exists \text{hasParent}.\text{Person} \sqsubseteq \text{Person}\}$
  - $\mathcal{A}_k = \{\text{hasParent}(a, a_1), \text{hasParent}(a_1, a_2), \ldots, \text{hasParent}(a_{k-1}, a_k), \text{Person}(a_k)\}$
  - $\langle \mathcal{T}, \mathcal{A}_k \rangle \models q$
  - $q_i = \exists x_1 \ldots x_i \text{hasParent}(a, x_1) \wedge \cdots \wedge \text{hasParent}(x_{i-1}, x_i), \text{Person}(x_i)$
  - $\mathcal{A}_k \models q_k$ but $\mathcal{A}_k \not\models q_i$ for $i \neq k$

# Conjunctive Query Answering in $\mathcal{EL}$

- In $\mathcal{EL}$, the chase does not terminate
  - $\text{Person} \sqsubseteq \exists \text{hasParent}.\text{Person}$
- The rewriting algorithm does not terminate either !
  - $\exists \text{hasParent}.\text{Person} \sqsubseteq \text{Person}$
- $\mathcal{EL}$ is not UCQ-rewritable
  - $q = \text{Person}(a)$
  - $\mathcal{T} = \{\exists \text{hasParent}.\text{Person} \sqsubseteq \text{Person}\}$
  - $\mathcal{A}_k = \{\text{hasParent}(a, a_1), \text{hasParent}(a_1, a_2), \ldots, \text{hasParent}(a_{k-1}, a_k), \text{Person}(a_k)\}$
  - $\langle \mathcal{T}, \mathcal{A}_k \rangle \models q$
  - $q_i = \exists x_1 \ldots x_i \text{hasParent}(a, x_1) \wedge \cdots \wedge \text{hasParent}(x_{i-1}, x_i), \text{Person}(x_i)$
  - $\mathcal{A}_k \models q_k$ but $\mathcal{A}_k \not\models q_i$ for $i \neq k$
- CQ answering in $\mathcal{EL}$ is still in $\text{PTIME}$
  - $\mathcal{EL}$ is Datalog-rewritable
  - also possible to combine some kind of chase and some FO-rewriting

# Tutorial: Analyzing Rule Sets with Kiabora

- A set of rules is a *finite expansion set* (fes) if its chase is finite.
- A set of rules is a *finite unification set* (fus) if every conjunctive query has a sound and complete UCQ-rewriting w.r.t. this set.

Deciding if a set of rules is a fus or a fes is undecidable in general but some recognizable rule classes are known to be fus or fes.

Kiabora is a tool dedicated to the analysis of a set of existential rules:

`https://graphik-team.github.io/graal/downloads/kiabora`