MAI4CAREU Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Ontological Query Answering

Andreas Pieris

Spring 2022-2023

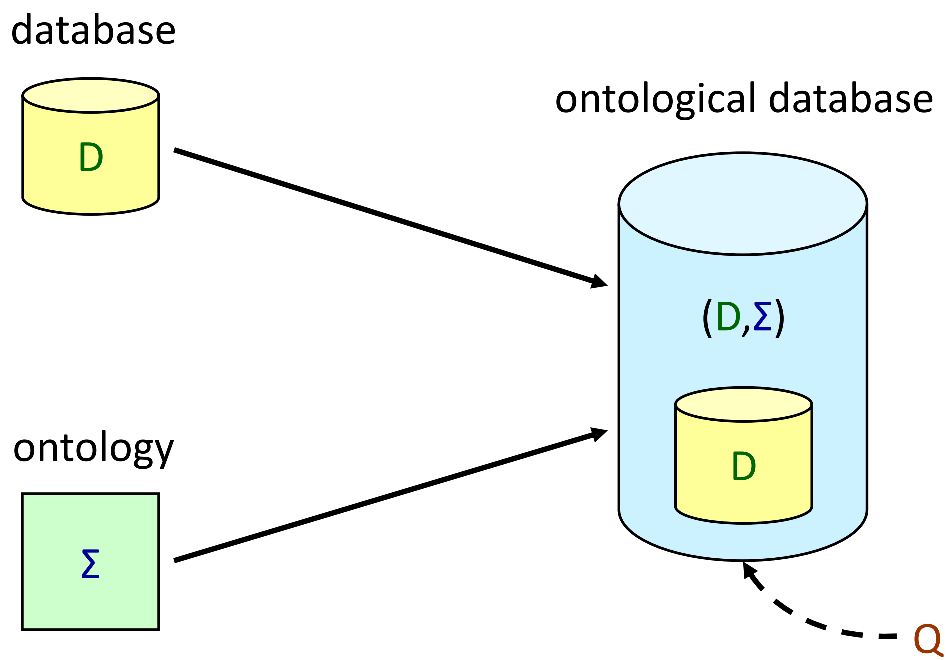
Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Learning Outcomes

- Ontological query answering via the chase procedure - forward-chaining
- Ontological query answering via query rewriting - backward-chaining
- Linear existential rules

Ontological Query Answering (OQA)



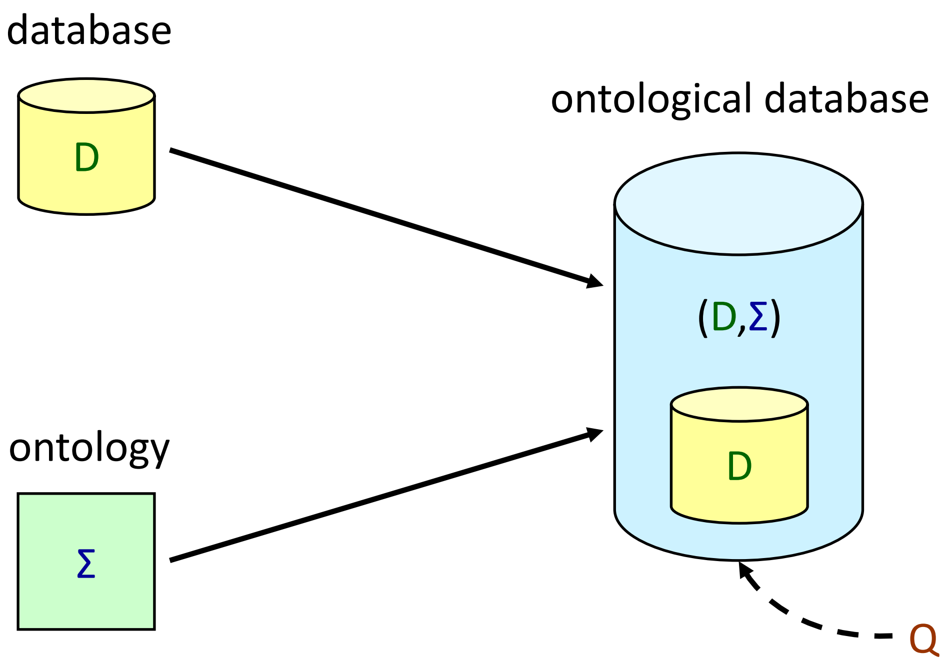
existential rules

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

conjunctive query

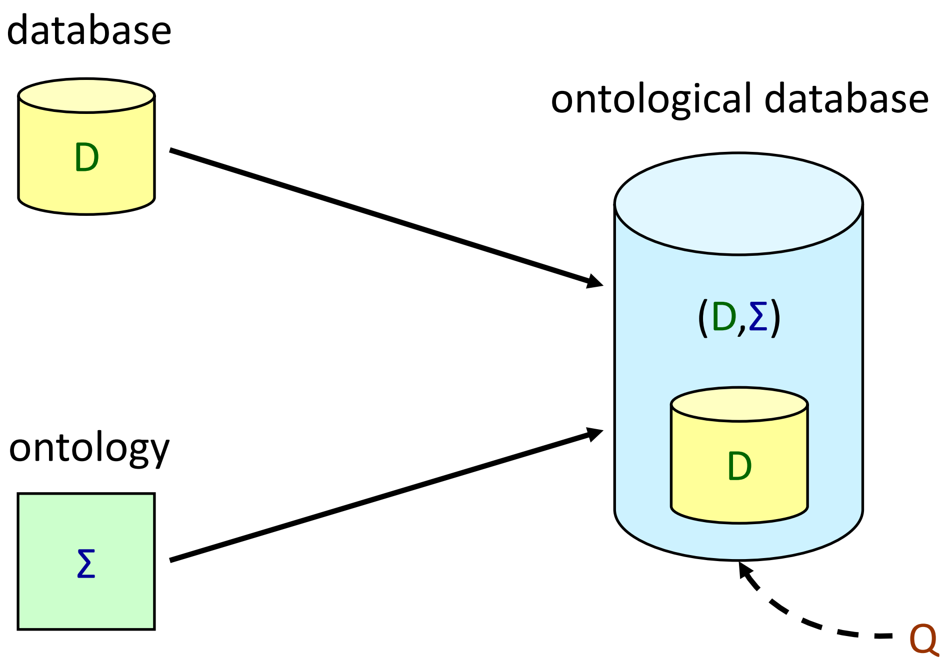
$Q(\mathbf{x})\ \ {:}{-}\ \ R_1(\mathbf{v_1}),\ldots,R_m(\mathbf{v_m})$

Ontological Query Answering (OQA)



$$\text{models}(D,\Sigma) \;=\; \{J \mid J \supseteq D \text{ and } J \models \Sigma\}$$

Ontological Query Answering (OQA)



$$\text{Answer}(Q,D,\Sigma) \quad = \quad \bigcap_{J \,\in\, \text{models}(D,\Sigma)} \quad Q(J)$$

Ontological Query Answering (OQA)

ontology language based on existential rules

OQA(**L**)

Input: a database D, a set of existential rules Σ ∈ **L**, a CQ Q/k, a tuple of constants **t** ∈ **adom**(D)^k

Question: **t** ∈ Answer(Q,D,Σ)?

BOQA(**L**)

Input: a database D, a set of existential rules Σ ∈ **L**, a Boolean query Q

Question: is Answer(Q,D,Σ) non-empty?

Theorem: OQA(**L**) $\equiv_L$ BOQA(**L**) for every language **L**

($\equiv_L$ means logspace-equivalent)

Data Complexity of BOQA

input D, fixed Σ and Q

BOQA[Σ,Q](**L**)

Input: a database D

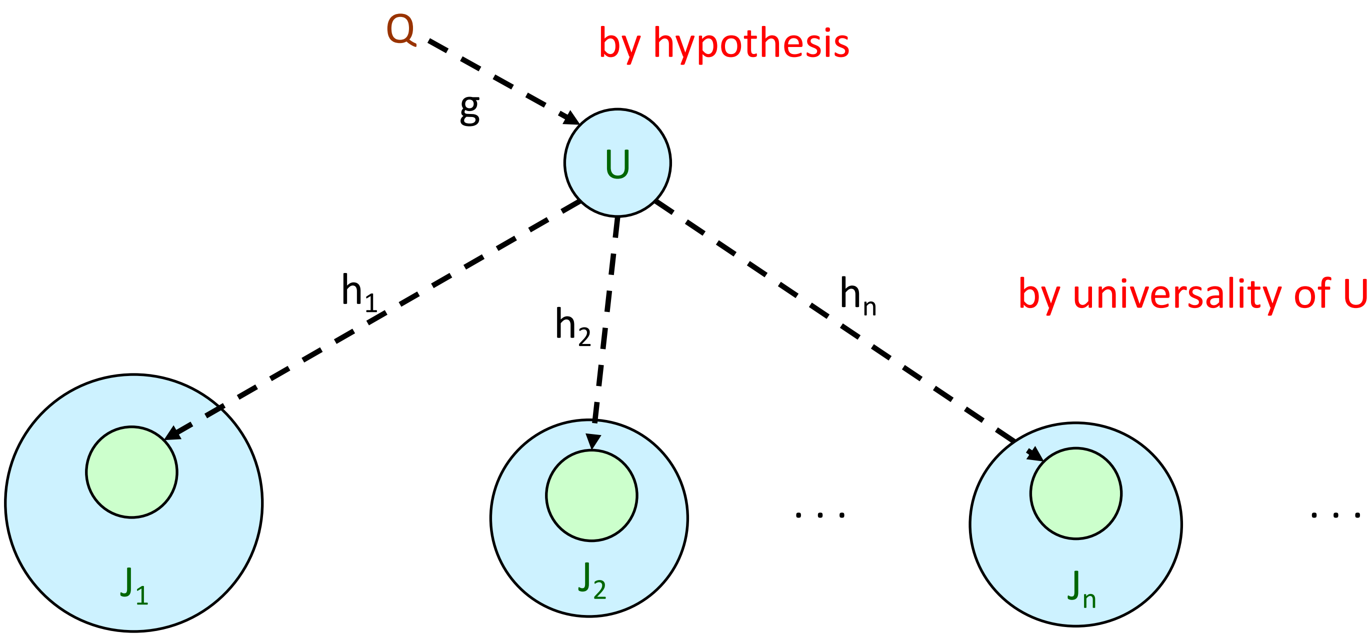
Question: is Answer(Q,D,Σ) non-empty?

Query Answering via Universal Models

Theorem: Answer(Q,D,Σ) is non-empty iff Q(U) is non-empty, where U a universal model of (D,Σ)

Proof: (⇒) Trivial since, for every J ∈ models(D,Σ), Q(J) is non-empty

(⇐) By exploiting the universality of U



∀J ∈ models(D,Σ), ∃h such that h(g(Q)) ⊆ J ⇒ ∀J ∈ models(D,Σ), Q(J) is non-empty

⇒ Answer(Q,D,Σ) is non-empty

The Chase Procedure

D

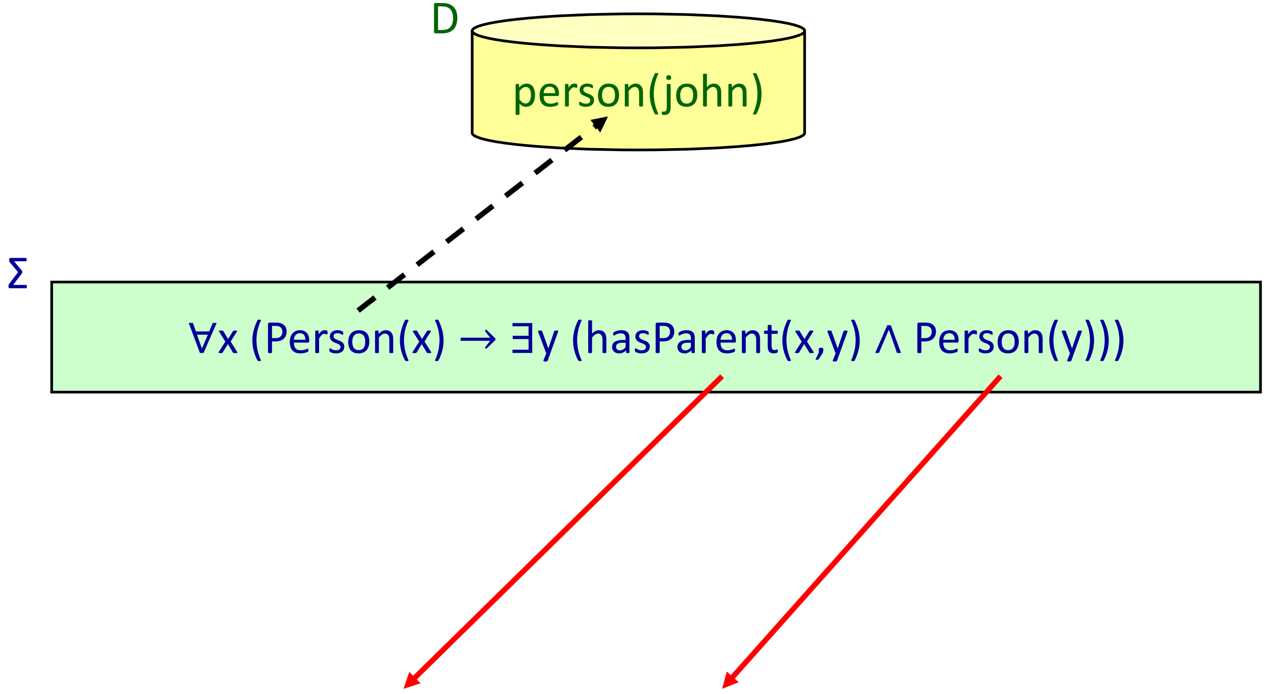
person(john)

Σ

$\forall x\,(Person(x) \rightarrow \exists y\,(hasParent(x,y) \wedge Person(y)))$

chase(D,Σ) = D ∪

The Chase Procedure



$\text{chase}(D,\Sigma) = D \cup \{\text{hasParent}(\text{john},\perp_1), \text{Person}(\perp_1)$

The Chase Procedure

D

person(john)

Σ

$\forall x\, (Person(x) \rightarrow \exists y\, (hasParent(x,y) \wedge Person(y)))$

$chase(D,\Sigma) = D \cup \{hasParent(john,\perp_1), Person(\perp_1),$

$hasParent(\perp_1,\perp_2), Person(\perp_2)$

The Chase Procedure

D

person(john)

Σ

$\forall x\,(\text{Person}(x) \rightarrow \exists y\,(\text{hasParent}(x,y) \wedge \text{Person}(y)))$

$\text{chase}(D,\Sigma) = D \cup \{\text{hasParent}(\text{john},\perp_1), \text{Person}(\perp_1),$

$\text{hasParent}(\perp_1,\perp_2), \text{Person}(\perp_2),$

$\text{hasParent}(\perp_2,\perp_3), \text{Person}(\perp_3)$

The Chase Procedure

D

person(john)

Σ

$\forall x\, (Person(x) \rightarrow \exists y\, (hasParent(x,y) \wedge Person(y)))$

$$chase(D,\Sigma) = D \cup \{hasParent(john,\perp_1),\ Person(\perp_1),$$

$$hasParent(\perp_1,\perp_2),\ Person(\perp_2),$$

$$hasParent(\perp_2,\perp_3),\ Person(\perp_3),\ \ldots$$

infinite instance

The Chase Procedure: Formal Definition

- Chase step - the building block of the chase procedure

- 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 an instance J if:

  1. There exists a homomorphism h such that $h(\varphi(\mathbf{x},\mathbf{y})) \subseteq J$
  2. There is no $g \supseteq h_{|\mathbf{x}}$ such that $g(\psi(\mathbf{x},\mathbf{z})) \subseteq J$

$J = \{R(a), P(a,b)\}$

$h = \{x \mapsto a\}$

$g = \{x \mapsto a, y \mapsto b\}$

$\forall x \, (R(x) \rightarrow \exists y \, P(x,y))$

✗

$J = \{R(a), P(b,a)\}$

$h = \{x \mapsto a\}$

✗

$\forall x \, (R(x) \rightarrow \exists y \, P(x,y))$

✓

The Chase Procedure: Formal Definition

- Chase step - the building block of the chase procedure

- 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 an instance J if:

    1. There exists a homomorphism h such that $h(\varphi(\mathbf{x},\mathbf{y})) \subseteq J$
    2. There is no $g \supseteq h_{|\mathbf{x}}$ such that $g(\psi(\mathbf{x},\mathbf{z})) \subseteq J$

- Let $J_+ = J \cup \{g(\psi(\mathbf{x},\mathbf{z}))\}$, where $g \supseteq h_{|\mathbf{x}}$ and $g(\mathbf{z})$ are "fresh" nulls not in J

- The result of applying σ to J is J_+, denoted $J[\sigma,h]J_+$ - single chase step

The Chase Procedure: Formal Definition

- A finite chase of D w.r.t. Σ is a finite sequence

$$D[\sigma_1,h_1]J_1[\sigma_2,h_2]J_2[\sigma_3,h_3]J_3 \cdots J_{n-1}[\sigma_n,h_n]J_n$$

and chase(D,Σ) is defined as the instance J_n

all applicable rules will eventually be applied

- An infinite chase of D w.r.t. Σ is a fair finite sequence

$$D[\sigma_1,h_1]J_1[\sigma_2,h_2]J_2[\sigma_3,h_3]J_3 \ldots J_{n-1}[\sigma_n,h_n]J_n \ldots$$

and chase(D,Σ) is defined as the instance $D \cup J_1 \cup J_2 \cup J_3 \cup \cdots \cup J_n \cup \cdots$

least fixpoint of a monotonic operator - chase step

Chase: A Universal Model

Theorem: chase(D,Σ) is a universal model of (D,Σ)

the result of the chase after $k \geq 0$ applications of the chase step

Proof:

- By construction, chase(D,Σ) ∈ models(D,Σ)
- It remains to show that chase(D,Σ) can be mapped into every other model of (D,Σ)
- Fix an arbitrary instance J ∈ models(D,Σ). We need to show that there exists h such that h(chase(D,Σ)) ⊆ J
- **By induction on the number of applications of the chase step, we show that for every $k \geq 0$, there exists h_k such that $h_k(\text{chase}^{[k]}(D,\Sigma)) \subseteq J$, and h_k is compatible with h_{k-1}**
- Clearly, $h_0 \cup h_1 \cup \cdots \cup h_n \cup \cdots$ is a well-defined homomorphism that maps chase(D,Σ) to J
- The claim follows with $h = h_0 \cup h_1 \cup \cdots \cup h_n \cup \cdots$

Chase: Uniqueness Property

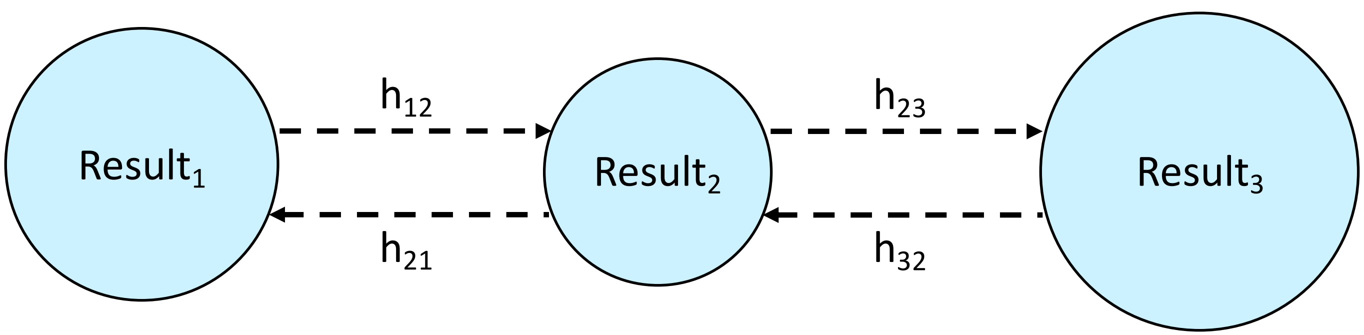
- The result of the chase is not unique - depends on the order of rule application

$D = \{P(a)\}$ $\sigma_1 = \forall x\,(P(x) \rightarrow \exists y\, R(y))$ $\sigma_2 = \forall x\,(P(x) \rightarrow R(x))$

$\text{Result}_1 = \{P(a), R(\perp), R(a)\}$ σ_1 then σ_2

$\text{Result}_2 = \{P(a), R(a)\}$ σ_2 then σ_1

- But, it is unique up to homomorphic equivalence



- Thus, it is unique for query answering purposes

Query Answering via the Chase

Theorem: Answer(Q,D,Σ) is non-empty iff Q(U) is non-empty, where U a universal model of (D,Σ)

&

Theorem: chase(D, Σ) is a universal model of (D,Σ)

⇓

Corollary: Answer(Q,D,Σ) is non-empty iff Q(chase(D,Σ)) is non-empty

- We can tame the first dimension of infinity by exploiting the chase procedure
- What about the second dimension of infinity? - the chase may be infinite

Can we tame the second dimension of infinity?

Undecidability of Ontological Query Answering

arbitrary existential rules

Theorem: OBQA(∃**RULES**) is undecidable

Proof Idea : By simulating a deterministic Turing machine with an empty tape.

Encode the computation of a DTM M with an empty tape using a database D, a set Σ of

existential rules, and a Boolean CQ Q such that Answer(Q,D,Σ) is non-empty iff M accepts

Gaining Decidability

By restricting the database

- Answer(Q,{Start(c)},Σ) is non-empty iff the DTM M accepts
- The problem is undecidable even for singleton databases
- No much to do in this direction

By restricting the query language

- Answer(Q :- Accept(x),D,Σ) is non-empty iff the DTM M accepts
- The problem is undecidable already for atomic queries
- No much to do in this direction

By restricting the ontology language

- Achieve a good trade-off between expressive power and complexity
- Field of intense research
- Any ideas?

Source of Non-termination

D

person(john)

Σ

$\forall x\, (Person(x) \rightarrow \exists y\, (hasParent(x,y) \wedge Person(y)))$

$chase(D,\Sigma) = D \cup \{hasParent(john,\perp_1), Person(\perp_1),$

$hasParent(\perp_1,\perp_2), Person(\perp_2),$

$hasParent(\perp_2,\perp_3), Person(\perp_3), \ldots$

1. **Existential quantification**
2. **Recursive definitions**

infinite instance

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**
  - Also known as non-recursive existential rules

Our Simple Example

D

person(john)

Σ

$\forall x\,(Person(x) \rightarrow \exists y\,(hasParent(x,y) \wedge Person(y)))$

$chase(D,\Sigma) = D \cup \{hasParent(john,\perp_1), Person(\perp_1),$

$hasParent(\perp_1,\perp_2), Person(\perp_2),$

$hasParent(\perp_2,\perp_3), Person(\perp_3), \ldots$

Existential quantification & recursive definitions

are key features for modelling ontologies

Key Question

We need classes of existential rules such that

- Existential quantification and recursive definition coexist

  $\Rightarrow$ the chase may be infinite

- BOQA is decidable, and tractable w.r.t. the data complexity

$\Downarrow$

Tame the infinite chase:

Deal with infinite structures without explicitly building them

Linear Existential Rules

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

$$\forall \mathbf{x} \forall \mathbf{y} \, (\mathrm{P}(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z} \, \psi(\mathbf{x},\mathbf{z}))$$

single atom

- We denote **LINEAR** the class of linear existential rules

- But, is this a reasonable ontology language?

3 OWL 2 QL

The OWL 2 QL profile is designed so that sound and complete query answering is in LOGSPACE (more precisely, in AC^0) with respect to the size of the data (assertions), while providing many of the main features necessary to express conceptual models such as UML class diagrams and ER diagrams. In particular, this profile contains the intersection of RDFS and OWL 2 DL. It is designed so that data (assertions) that is stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism, i.e., by rewriting the query into an SQL query that is then answered by the RDBMS system, without any changes to the data.

OWL 2 QL is based on the DL-Lite family of description logics [*DL-Lite*]. Several variants of DL-Lite have been described in the literature, and DL-Lite$_R$ provides the logical underpinning for OWL 2 QL. DL-Lite$_R$ does not require the unique name assumption (UNA), since making this assumption would have no impact on the semantic consequences of a DL-Lite$_R$ ontology. More expressive variants of DL-Lite, such as DL-Lite$_A$, extend DL-Lite$_R$ with functional properties, and these can also be extended with keys; however, for query answering to remain in LOGSPACE, these extensions require UNA and need to impose certain global restrictions on the interaction between properties used in different types of axiom. Basing OWL 2 QL on DL-Lite$_R$ avoids practical problems involved in the explicit axiomatization of UNA. Other variants of DL-Lite can also be supported on top of OWL 2 QL, but may require additional restrictions on the structure of ontologies.

3.1 Feature Overview

OWL 2 QL is defined not only in terms of the set of supported constructs, but it also restricts the places in which these constructs are allowed to occur. The allowed usage of constructs in class expressions is summarized in Table 1.

Table 1. Syntactic Restrictions on Class Expressions in OWL 2 QL

Subclass Expressions	Superclass Expressions
a class existential quantification (**ObjectSomeValuesFrom**) where the class is limited to *owl:Thing* existential quantification to a data range (**DataSomeValuesFrom**)	a class intersection (**ObjectIntersectionOf**) negation (**ObjectComplementOf**) existential quantification to a class (**ObjectSomeValuesFrom**) existential quantification to a data range (**DataSomeValuesFrom**)

OWL 2 QL supports the following axioms, constrained so as to be compliant with the mentioned restrictions on class expressions:

- subclass axioms (**SubClassOf**)
- class expression equivalence (**EquivalentClasses**)
- class expression disjointness (**DisjointClasses**)
- inverse object properties (**InverseObjectProperties**)
- property inclusion (**SubObjectPropertyOf** not involving property chains and **SubDataPropertyOf**)
- property equivalence (**EquivalentObjectProperties** and **EquivalentDataProperties**)
- property domain (**ObjectPropertyDomain** and **DataPropertyDomain**)
- property range (**ObjectPropertyRange** and **DataPropertyRange**)
- disjoint properties (**DisjointObjectProperties** and **DisjointDataProperties**)

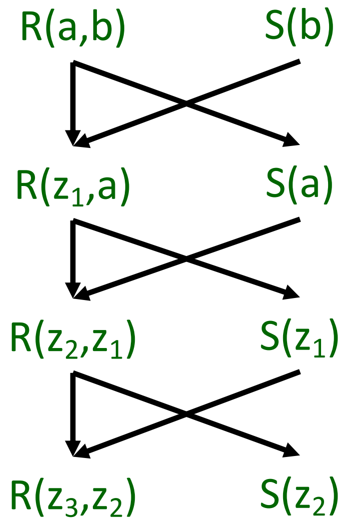
https://www.w3.org/TR/owl2-profiles/#OWL_2_QL

Chase Graph

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

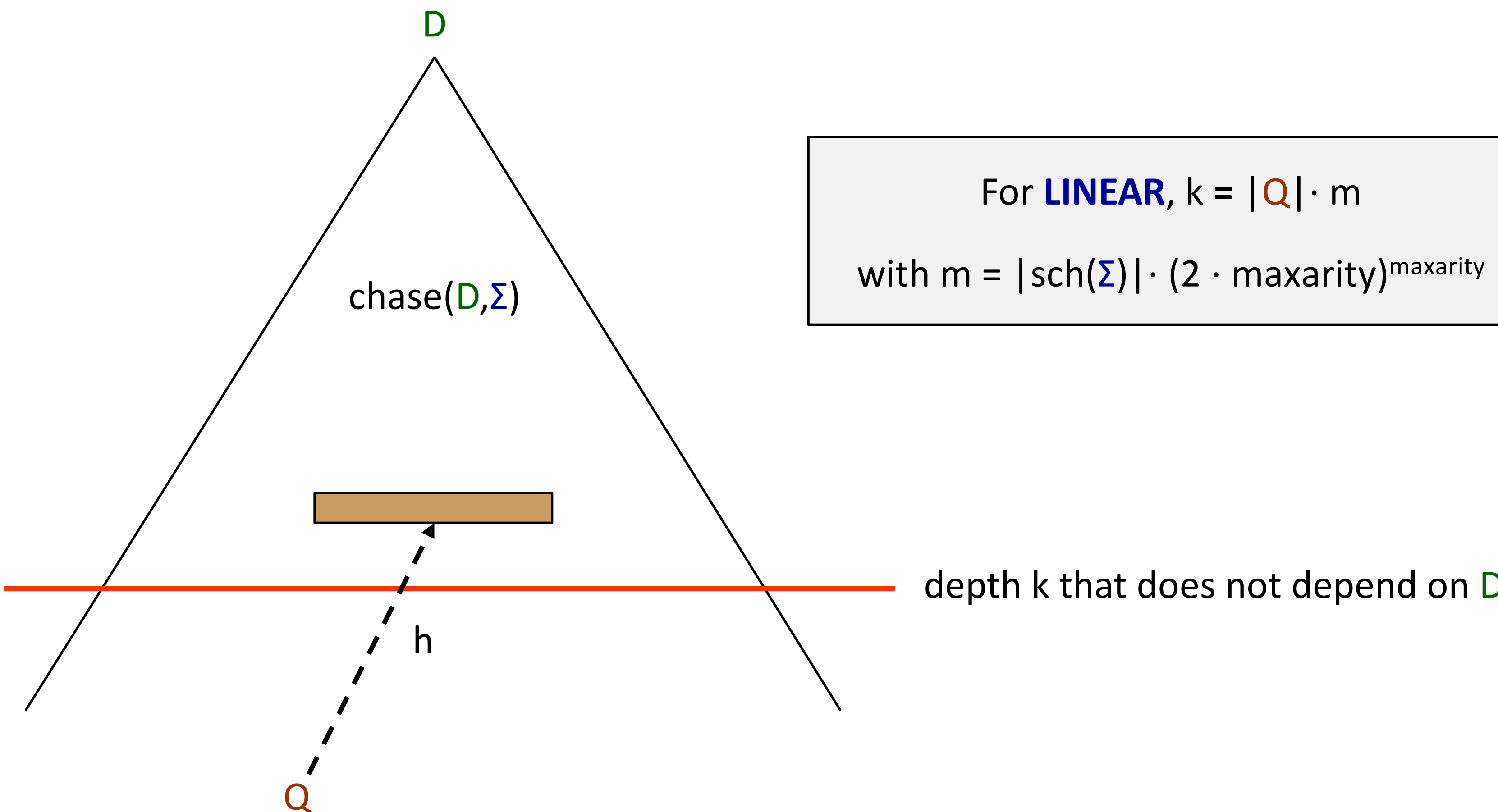
$D = \{R(a,b), S(b)\}$

$$\Sigma = \begin{cases} \forall x \forall y \, (R(x,y) \wedge S(y) \rightarrow \exists z \, R(z,x)) \\ \forall x \forall y \, (R(x,y) \rightarrow S(x)) \end{cases}$$



For **LINEAR** the chase graph is a forest

Bounded Derivation-Depth Property



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

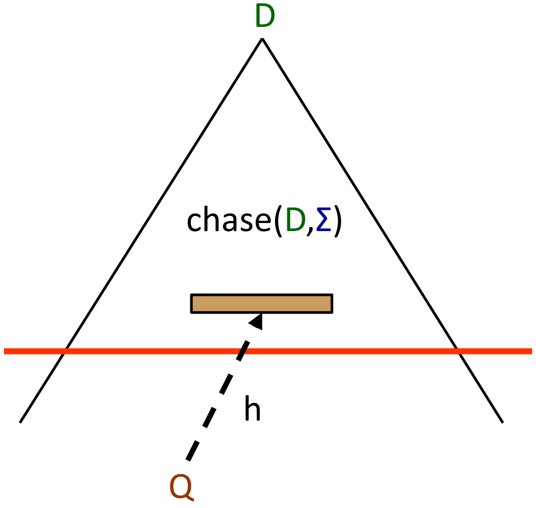
with $m = |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$

chase graph up to depth k

$Q(chase(D,\Sigma))$ is non-empty $\Rightarrow$ $Q(chase^k(D,\Sigma))$ is non-empty

The Blocking Algorithm for **LINEAR**

Theorem: BOQA[Σ,Q](**LINEAR**) is in PTIME for a fixed set Σ, and a Boolean CQ Q



$k = |Q| \cdot |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$

The Blocking Algorithm for **LINEAR**

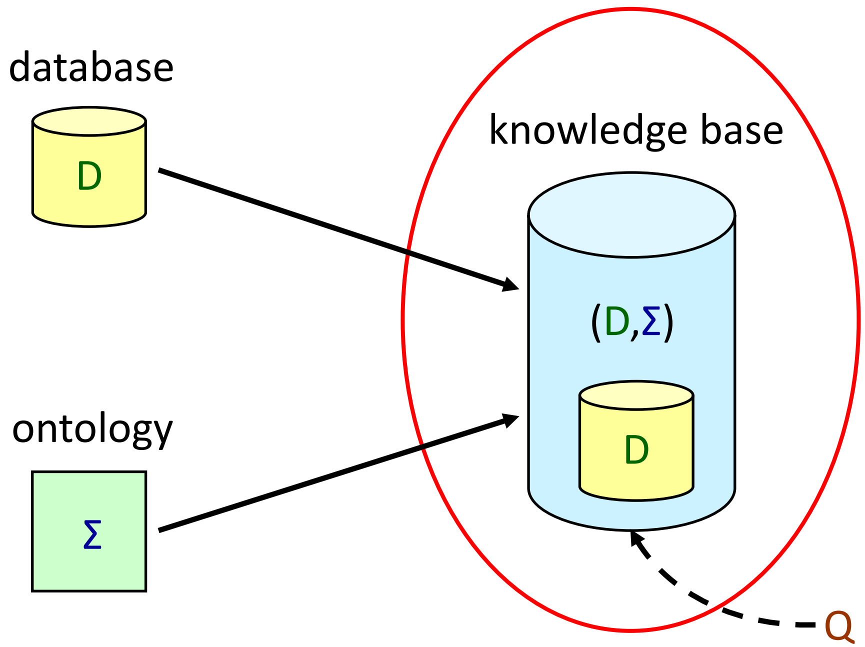
Theorem: BOQA[Σ,Q](**LINEAR**) is in PTIME for a fixed set Σ, and a Boolean CQ Q

but, we can do better

Theorem: BOQA[Σ,Q](**LINEAR**) is in LOGSPACE for a fixed set Σ, and a Boolean CQ Q

Scalability in OQA

Exploit standard RDBMSs - efficient technology for answering CQs



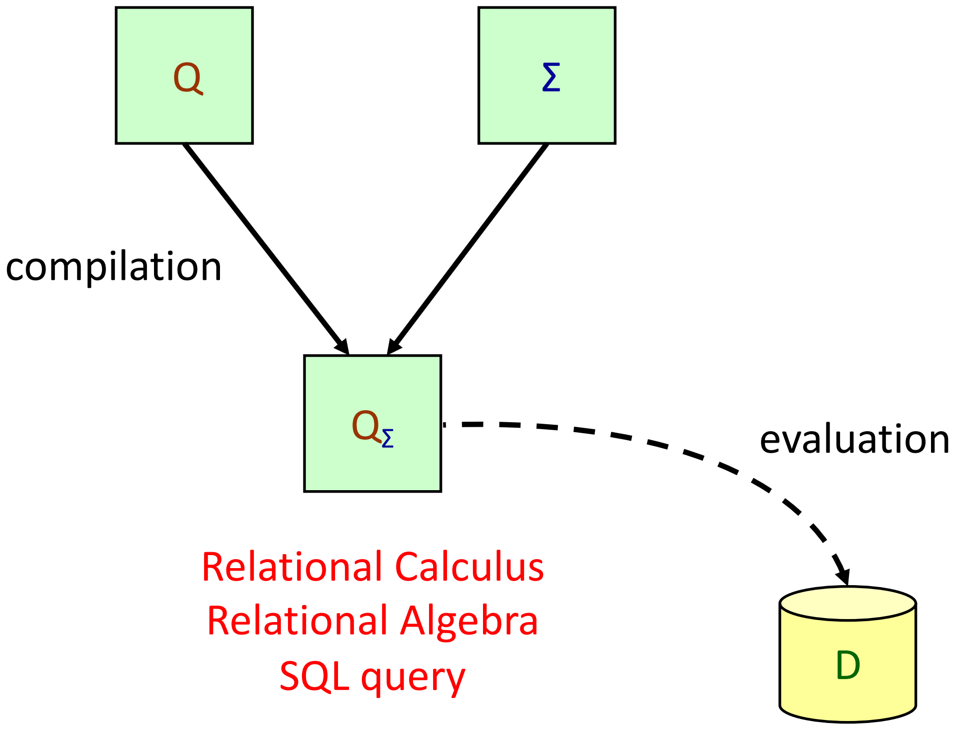
But in the OQA setting

we have to query a

knowledge base, not just a

relational database

Query Rewriting



for every database D, Answer(Q,D,Σ) is non-empty iff Q_Σ(D) is non-empty

Query Rewriting: Formal Definition

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

BOQA(**L**) is **Q**-rewritable if, for every $\Sigma \in$ **L** and Boolean CQ Q,

we can construct a Boolean query $Q_\Sigma \in$ **Q** such that,

for every database D, Answer(Q,D,Σ) is non-empty iff Q_Σ(D) is non-empty

NOTE: The construction of Q_Σ is database-independent

An Example

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

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

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

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

Q_2 :- $R(x,z), U(x,y), T(y),$

Q_3 :- $R(x,z), U(x,y), P(y)\}$

An Example

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

Q :- $P(c)$

$$
\begin{aligned}
Q_\Sigma = \{ & Q \text{ :- } P(c), \\
& Q_1 \text{ :- } R(c,y_1),\ P(y_1), \\
& Q_2 \text{ :- } R(c,y_1),\ R(y_1,y_2),\ P(y_2), \\
& Q_3 \text{ :- } R(c,y_1),\ R(y_1,y_2),\ R(y_2,y_3),\ P(y_3), \\
& \ldots \}
\end{aligned}
$$

- This cannot be written as a finite first-order query
- It can be written as Q :- $R(c,x)$, $R^*(x,y)$, $P(y)$, but transitive closure is not FO-expressible

Query Rewriting for **LINEAR**

union of conjunctive queries

Theorem: **LINEAR** is UCQ-rewritable

$\Downarrow$

Theorem: BOQA[Σ,Q](**LINEAR**) is in LOGSPACE for a fixed set Σ, and a Boolean CQ Q

…it also tells us that for answering CQs in the presence of **LINEAR** ontologies,

we can exploit standard database technology

UCQ-Rewritings

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

    1. Rewriting

    2. Minimization

- We are going to see the version of the algorithm that assumes normalized existential rules, where only one atom appears in the head

Normalization Procedure

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

$$\downarrow$$

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

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

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

...

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

NOTE : Linearity is preserved, and we obtain an equivalent ontology w.r.t. query answering

UCQ-Rewritings

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

- We are going to see the version of the algorithm that assumes normalized existential rules, where only one atom appears in the head

Rewriting Step

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

Q :- hasCollaborator(u,db,v)

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

hasCollaborator(u,db,v)

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

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

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

Unsound Rewritings

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

Q :- $hasCollaborator(c,db,v)$

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

(c is a constant)

$hasCollaborator(c,db,v)$

After applying the rewriting step we obtain the following UCQ

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

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

Unsound Rewritings

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

Q :- hasCollaborator(c,db,v)

Q_Σ = {Q :- hasCollaborator(c,db,v),

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

- Consider the database D = {project(a), inArea(a,db)}
- Clearly, Q_Σ(D) is non-empty
- However, Answer(Q,D,Σ) is empty since there is no way to obtain an atom of the form hasCollaborator(c,db,_) during the chase

Unsound Rewritings

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

Q :- $hasCollaborator(c,db,v)$

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

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

the information about the constant c in the original query is lost after the application of the rewriting step since c is unified with an ∃-variable

Unsound Rewritings

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

Q :- hasCollaborator(v,db,v)

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

hasCollaborator(v,db,v)

After applying the rewriting step we obtain the following UCQ

Q_Σ = {Q :- hasCollaborator(v,db,v),

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

Unsound Rewritings

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

Q :- hasCollaborator(v,db,v)

$$\boxed{\begin{aligned} Q_\Sigma = \{ & Q \text{ :- hasCollaborator(v,db,v)}, \\ & Q_1 \text{ :- project(v), inArea(v,db)}\} \end{aligned}}$$

- Consider the database D = {project(a), inArea(a,db)}
- Clearly, $Q_\Sigma(D)$ is non-empty
- However, Answer(Q,D,Σ) is empty since there is no way to obtain an atom of the form hasCollaborator(t,db,t) during the chase

Unsound Rewritings

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

Q :- $hasCollaborator(v,db,v)$

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

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

Applicability Condition

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

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

1. head(σ) and α unify via h

2. For every variable x in head(σ):
    1. If h(x) is a constant, then x is a ∀-variable
    2. If h(x) = h(y), where y is a shared variable of α, then x is a ∀-variable

3. If x is an ∃-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 it is crucial for soundness, may destroy completeness

Incomplete Rewritings

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

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

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

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

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

- Consider the database D = {project(a), inArea(a,db)}
- Clearly, Q over chase(D,Σ) = D ∪ {hasCollaborator(z,db,a), collaborator(z)} is non-empty
- However, Q_Σ(D) is empty

Incomplete Rewritings

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

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

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

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

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

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

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

Incomplete Rewritings

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

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

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

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

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

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

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

$Q_\Sigma(D)$ is non-empty, where $D = \{project(a), inArea(a,db)\}$

UCQ-Rewritings

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

    1. Rewriting

    2. Minimization

- We are going to see the version of the algorithm that assumes 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 := $Q_\Sigma \cup \{q_{rew}\}$
    //Minimization Step
        foreach pair of atoms α,β in q that <u>unify</u> **do**
            q_{min} := minimize(q,α,β)   //we apply the most general unifier of α and β on q
            if q_{min} does not appear in Q_Σ (modulo variable renaming) **then**
                        Q_Σ := $Q_\Sigma \cup \{q_{min}\}$
until $Q_{aux} = Q_\Sigma$
return Q_Σ

Termination

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

Proof Idea:

- **Key observation:** the size of each partial rewriting is at most the size of the given CQ Q
- Thus, each partial rewriting can be transformed into an equivalent query that contains at most ($|Q| \cdot$ maxarity) variables
- The number of queries that can be constructed using a finite number of predicates and a finite number of variables is finite
- Therefore, only finitely many partial rewritings can be constructed - in general, exponentially many

Size of the Rewriting

- Ideally, we would like to construct UCQ-rewritings of polynomial size
- But, the standard rewritng algorithm produces rewritings of exponential size
- Can we do better? **NO!!!**

$\Sigma = \{\forall x\, (R_k(x) \rightarrow P_k(x))\}$ for $k \in \{1,\ldots,n\}$ $\quad Q \text{ :- } P_1(x), \ldots, P_n(x)$

$Q \text{ :- } P_1(X), \ldots, P_n(X)$

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

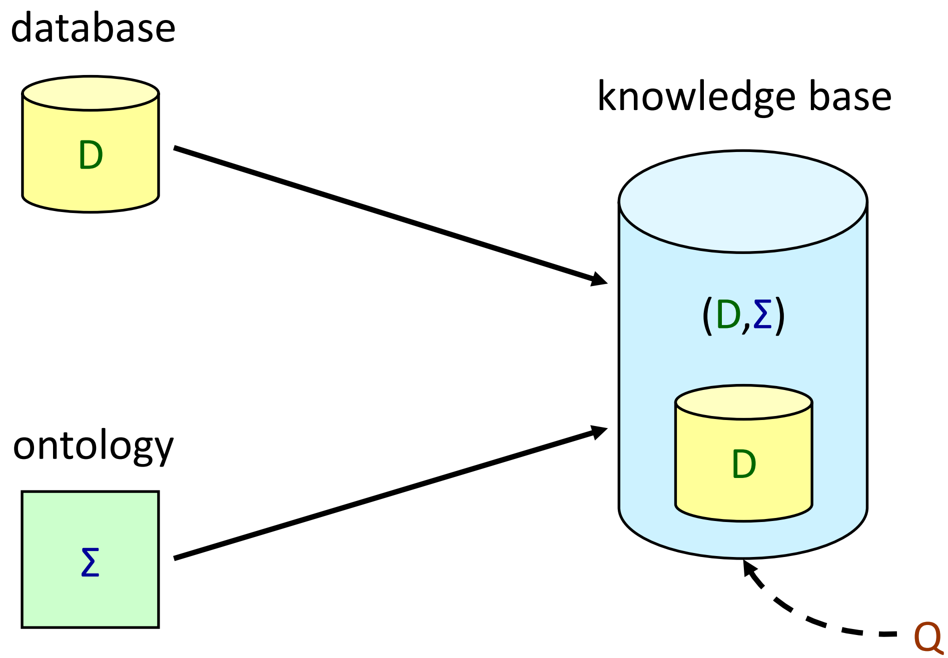
thus, we need to consider 2^n disjuncts

Size of the Rewriting

- Ideally, we would like to construct UCQ-rewritings of polynomial size
- But, the standard rewritng algorithm produces rewritings of exponential size
- Can we do better? **NO!!!**

  - Although the standard rewriting algorithm is worst-case optimal, it can be significantly improved
  - Optimization techniques can be applied in order to compute efficiently small rewritings - field of intense research

Recap



existential rules

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

conjunctive query

$Q(\mathbf{x}) \;\; \text{:-} \;\; R_1(\mathbf{v_1}),\ldots,R_m(\mathbf{v_m})$

in general, this is an undecidable problem, but well-behaved ontology languages exists - **LINEAR**

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Thank You!

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423