Ontologies in Computer Science: Principles, Methods, and Applications to Data Management

Maurizio Lenzerini

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

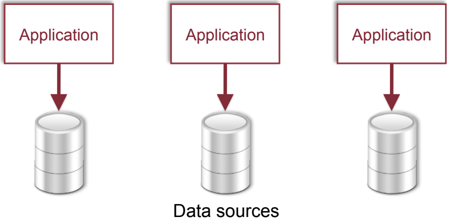
SAPIENZA
UNIVERSITÀ DI ROMA

Part I

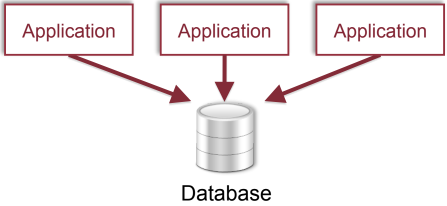
Seminars in Advanced Topics in Computer Science Engineering
April 27 - May 4, 2018

Information system architecture enabled by DBMS

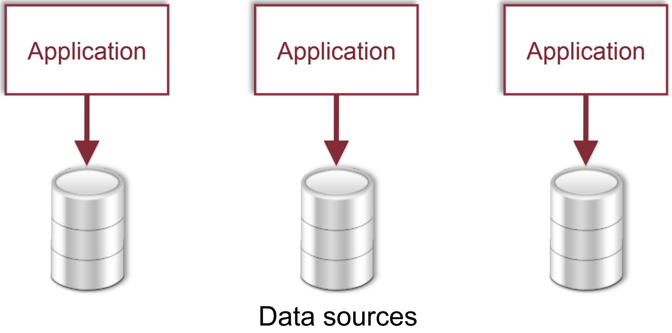
Pre-DBMS architecture (need of a unified data storage):



"Ideal information system architecture" with DBMS ('70s):



Today in many organizations ...



- Distributed, redundant, application-dependent, and mutually incoherent data
- Desperate need of a coherent, conceptual, unified view of data

Fragment of a relational table in a Bank Information system:

CUC	TS_START	TS_END	ID_GRUP	FLAG_CP	FLAG_CF	FATTURATO	FLAG_FATT
124589	30-lug-2004	1-gen-9999	92736	S	N	195000,00	N
140904	15-mag-2001	15-giu-2005	35060	N	N	230600,00	N
124589	5-mag-2001	30-lug-2004	92736	N	S	195000,00	S
-452901	13-mag-2001	27-lug-2004	92770	S	N	392000,00	N
129008	10-mag-2001	1-gen-9999	62010	N	S	247000,00	S
-472900	10-mag-2001	1-gen-9999	62010	S	N	0 00	N
130976	7-mag-2001	9-lug-2003	75680				

... even with just one data source

Negative value denotes a holding

CUC	TS_START	TS_END	ID_GRUP	FLAG_CP	FLAG_CF	FATTURATO	FLAG_FATT	
124589	30-lug-2004	1-gen-9999	92736	S	N	195000,00	N	
140904	15-mag-2001	15-giu-2005	35060	N	N	230600,00	N	
124589	5-mag-2001	30-lug-2004	92736	N	S	195000,00	S	
-452901	13-mag-2001	27-lug-2004	92770	S	N	392000,00	N	
129008	10-mag-2001	1-gen-9999	62010	N	S	247000,00	S	
-472900	10-mag-2001	1-gen-9999	62010	S	N	0 00	N	
130976	7-mag-2001	9-lug-2003	75680					

S means that the customer is the leader of the group it belongs to

S means that the customer is the head of the group it belongs to

CUC	TS_START	TS_END	ID_GRUP	FLAG_CP	FLAG_CF	FATTURATO	FLAG_FATT
124589	30-lug-2004	1-gen-9999	92736	S	N	195000,00	N
140904	15-mag-2001	15-giu-2005	35060	N	N	230600,00	N
124589	5-mag-2001	30-lug-2004	92736	N	S	195000,00	S
-452901	13-mag-2001	27-lug-2004	92770	S	N	392000,00	N
129008	10-mag-2001	1-gen-9999	62010	N	S	247000,00	S
-472900	10-mag-2001	1-gen-9999	62010	S	N	0 00	N
130976	7-mag-2001	9-lug-2003	75680				

… even with just one data source

N means that the FATTURATO field is not valid

CUC	TS_START	TS_END	ID_GRUP	FLAG_CP	FLAG_CF	FATTURATO	FLAG_FATT	
124589	30-lug-2004	1-gen-9999	92736	S	N	195000,00	N	
140904	15-mag-2001	15-giu-2005	35060	N	N	230600,00	N	
124589	5-mag-2001	30-lug-2004	92736	N	S	195000,00	S	
-452901	13-mag-2001	27-lug-2004	92770	S	N	392000,00	N	
129008	10-mag-2001	1-gen-9999	62010	N	S	247000,00	S	
-472900	10-mag-2001	1-gen-9999	62010	S	N	0 00	N	
130976	7-mag-2001	9-lug-2003	75680					

Data preparation and information integration

- Large enterprises spend a great deal of time and money on data preparation and information integration (∼40% of information-technology shops' budget).
- Market for information integration software estimated to grow to $3.4 billion by 2019 [Gartner, 2015]
- Data integration is a large and growing part of software development, computer science, and specific applications settings, such as scientific computing, semantic web, etc..
- Data preparation and integration is crucial for "big data" processing (to make sense of big data!)

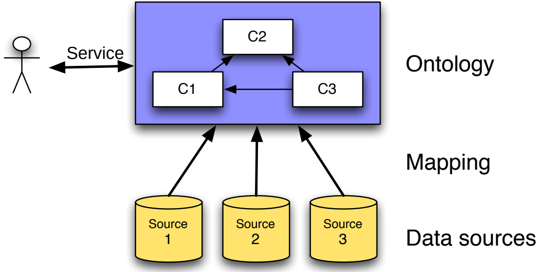
Basing the integrated view of data on a clean, rich and abstract conceptual representation of the data has always been both a goal and a challenge [Mylopoulos et al 1984]

Managing data through the lens of an ontology: Ontology-based Data Management

Ontology-based Data Management is a new paradigm, rooted on the idea of using Database Theory fundamentals, and Logic-based Knowledge Representation and Reasoning techniques for a new way of managing data, and characterized by the following principles:

- Data may reside where they are (no need to move data)
- Build a conceptual specification of the domain of interest, in terms of knowledge structures
- Map such knowledge structures to concrete data sources
- Express all services over the knowledge structures
- Automatically translate knowledge services to data services

Ontology-based data management: architecture



Based on three main components:

- Ontology, a declarative, logic-based specification of the domain of interest, used as a unified, conceptual view for clients
- Data sources, representing external, independent, heterogeneous, storage (or, more generally, computational) structures
- Mappings, used to semantically link data at the sources to the ontology

- Part I
  - Ontology-based data management: The framework
  - Queries in OBDM
  - The nature of query answering in OBDM
- Part II
  - Ontology languages
  - Modeling the domain through the ontology
  - Modeling the mapping with the data sources
- Part III
  - Algorithms for query answering
  - Beyond classical first-order queries

Outline of part I

1. Ontology-based data management: The framework
2. Queries in OBDM
3. The nature of query answering in OBDM

Formal framework of ontology-based data management

An ontology-based data management (OBDM or OBDA) system is a triple $\Sigma = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{O}$ is the ontology, expressed as a logical theory (here, a TBox in a Description Logic)
- $\mathcal{S}$ is a relational database representing the data sources (note that federation tools are able to present a set of heterogeneous data sources as a single relational database)
- $\mathcal{M}$ is a set of mapping assertions, each one of the form

$$\Phi(\vec{x}) \leadsto \Psi(\vec{x})$$

  where

  - $\Phi(\vec{x})$ is a FOL query over $\mathcal{S}$, returning values for $\vec{x}$
  - $\Psi(\vec{x})$ is a FOL query over $\mathcal{O}$, whose free variables are from $\vec{x}$.

Ontology-based data management system – Example

Ontology $\mathcal{O}$

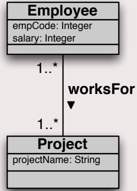
DL notation:	Classical FOL notation:
$\text{Employee} \sqsubseteq \exists\text{worksFor}$	$\forall x\, \text{Employee}(x) \rightarrow \exists y\, \text{worksFor}(x,y)$
$\text{Employee} \sqsubseteq \exists\text{empCode}$	$\forall x\, \text{Employee}(x) \rightarrow \exists y\, \text{empCode}(x,y)$
$\text{Employee} \sqsubseteq \exists\text{salary}$	$\forall x\, \text{Employee}(x) \rightarrow \exists y\, \text{salary}(x,y)$
$\text{Project} \sqsubseteq \exists\text{worksFor}^-$	$\forall x\, \text{Project}(x) \rightarrow \exists y\, \text{worksFor}(y,x)$
$\text{Project} \sqsubseteq \exists\text{projectName}$	$\forall x\, \text{Project}(x) \rightarrow \exists y\, \text{projectName}(x,y)$
$\exists\text{worksFor} \sqsubseteq \text{Employee}$	$\forall x \forall y\, \text{worksFor}(x,y) \rightarrow \text{Employee}(x)$
$\exists\text{worksFor}^- \sqsubseteq \text{Project}$	$\forall x \forall y\, \text{worksFor}(x,y) \rightarrow \text{Project}(y)$

- DLs use unary predicates (concepts, or classes), and binary predicates between classes (relations, or roles, or object properties), and other binary predicates relating classes to value types (attributes, or data properties)
- $\rightarrow$ corresponds to $\sqsubseteq$
- R^- denotes the inverse of the relation R
- $\lambda x.C(x)$ is written as C
- $\lambda x.\exists y R(x,y)$ is written as $\exists R$

Ontology-based data management system – Example

Ontology $\mathcal{O}$ (TBox)

$\text{Employee} \sqsubseteq \exists\text{worksFor}$
$\text{Employee} \sqsubseteq \exists\text{empCode}$
$\text{Employee} \sqsubseteq \exists\text{salary}$
$\text{Project} \sqsubseteq \exists\text{worksFor}^-$
$\text{Project} \sqsubseteq \exists\text{projectName}$
$\exists\text{worksFor} \sqsubseteq \text{Employee}$
$\exists\text{worksFor}^- \sqsubseteq \text{Project}$



Federated schema of the DB $\mathcal{S}$

D_1 [*SSN*: String, *PrName*: String]
Employees and Projects they work for

D_2 [*Code*: String, *Salary*: Int]
Employee's Code with salary

D_3 [*Code*: String, *SSN*: String]
Employee's Code with SSN

...

Mapping $\mathcal{M}$

M_1: `SELECT SSN,PrName FROM D1` $\rightsquigarrow$ V_1(SSN,PrName) $\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), workFor(**pers**(*SSN*), **proj**(*PrName*))

M_2: `SELECT SSN,Salary FROM D2, D3 WHERE D2.Code = D3.Code` $\rightsquigarrow$ V_2(SSN,Salary) $\rightsquigarrow$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

Note: in practice we often write mappings using an intermediate view symbol.

Semantics

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation for the ontology $\mathcal{O}$, where $\Delta^{\mathcal{I}}$ is the domain and $\cdot^{\mathcal{I}}$ is the interpretation function.

Def.: Mapping satisfaction (sound mappings)

We say that $\mathcal{I}$ satisfies $\Phi(\vec{x}) \leadsto \Psi(\vec{x})$ wrt a database $\mathcal{S}$, if the sentence

$$\forall \vec{x}\ (\Phi(\vec{x}) \rightarrow \Psi(\vec{x}))$$

is true in $\mathcal{I} \cup \mathcal{S}$.

Def.: Model

$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of $\Sigma = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ if:

- $\mathcal{I}$ is a model of $\mathcal{O}$, i.e., it satisfies all axioms in $\mathcal{O}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ wrt $\mathcal{S}$, i.e., satisfies every assertion in $\mathcal{M}$ wrt $\mathcal{S}$.

Def.: Semantics

The semantics of Σ is the set $sem(\Sigma)$ of all models of Σ.

Ontology-based data management (OBDM): topics

- *Ontology-based [data access | query answering] (OBDA | OBQA)*
- *Ontology-based data quality (OBDQ)*
- *Ontology-based data governance (OBDG)*
- *Ontology-based data restructuring (OBDR)*
- *Ontology-based business intelligence (OBBI)*
- *Ontology-based data exchange and coordination (OBDE)*
- *Ontology-based data update (OBDU)*
- *Ontology-based service and process management (OBDS)*

General requirements:

- large data collections
- efficiency with respect to size of data (data complexity)

Outline of part I

Conjunctive queries

- are the most common kind of first-order queries
- also known as select-project-join SQL queries
- allow for easy optimization in relational DBMSs

Definition

A conjunctive query (CQ) is a first-order query of the form

$$\{ \ (\vec{\mathbf{x}}) \ \mid \ \exists \vec{\mathbf{y}}.\ r_1(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1) \wedge \cdots \wedge r_m(\vec{\mathbf{x}}_m, \vec{\mathbf{y}}_m) \ \}$$

where

- $\vec{\mathbf{x}}$ is the union of the $\vec{\mathbf{x}}_i$'s, and $\vec{\mathbf{y}}$ is the union of the $\vec{\mathbf{y}}_i$'s
- $r_1, \ldots, r_m$ are relation symbols (not built-in predicates)

We use the following abbreviation: $\{ \ (\vec{\mathbf{x}}) \mid r_1(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), \ldots, r_m(\vec{\mathbf{x}}_m, \vec{\mathbf{y}}_m) \ \}$

SAPIENZA UNIVERSITÀ DI ROMA

Conjunctive queries

- are the most common kind of first-order queries
- also known as select-project-join SQL queries
- allow for easy optimization in relational DBMSs

Definition

A conjunctive query (CQ) is a first-order query of the form

$$\{ \ (\vec{\mathbf{x}}) \ \mid \ \exists \vec{\mathbf{y}}.\ r_1(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1) \wedge \cdots \wedge r_m(\vec{\mathbf{x}}_m, \vec{\mathbf{y}}_m) \ \}$$

where

- $\vec{\mathbf{x}}$ is the union of the $\vec{\mathbf{x}}_i$'s, and $\vec{\mathbf{y}}$ is the union of the $\vec{\mathbf{y}}_i$'s
- $r_1, \ldots, r_m$ are relation symbols (not built-in predicates)

We use the following abbreviation: $\{ \ (\vec{\mathbf{x}}) \mid r_1(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), \ldots, r_m(\vec{\mathbf{x}}_m, \vec{\mathbf{y}}_m) \ \}$

Complexity of relational calculus

We consider the complexity of the recognition problem, i.e., checking whether a tuple of constants is in the answer to a query:

- measured wrt the size of the database $\rightsquigarrow$ data complexity
- measured wrt the size of the query and the database $\rightsquigarrow$ combined complexity

Complexity of relational calculus

- data complexity: polynomial, actually in LogSpace (or, in terms of circuit complexity, in AC_0)
- combined complexity: PSpace-complete

Complexity of conjunctive queries

- data complexity: in LogSpace
- combined complexity: NP-complete

Complexity of relational calculus

We consider the complexity of the recognition problem, i.e., checking whether a tuple of constants is in the answer to a query:

- measured wrt the size of the database $\rightsquigarrow$ data complexity
- measured wrt the size of the query and the database $\rightsquigarrow$ combined complexity

Complexity of relational calculus

- data complexity: polynomial, actually in LogSpace (or, in terms of circuit complexity, in AC_0)
- combined complexity: PSpace-complete

Complexity of conjunctive queries

- data complexity: in LogSpace
- combined complexity: NP-complete

Complexity of relational calculus

We consider the complexity of the recognition problem, i.e., checking whether a tuple of constants is in the answer to a query:

- measured wrt the size of the database $\rightsquigarrow$ data complexity
- measured wrt the size of the query and the database $\rightsquigarrow$ combined complexity

Complexity of relational calculus

- data complexity: polynomial, actually in $\textsc{LogSpace}$ (or, in terms of circuit complexity, in AC_0)
- combined complexity: $\textsc{PSpace}$-complete

Complexity of conjunctive queries

- data complexity: in $\textsc{LogSpace}$
- combined complexity: NP-complete

Queries in OBDM

- The domain Δ is fixed, and we do not distinguish an element of Δ from the constant denoting it $\rightsquigarrow$ standard names
- Queries to $\Sigma = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ are first-order queries over the alphabet $\mathcal{A}_{\mathcal{O}}$ of the ontology
- When "evaluating" q over Σ, we have to consider that there may be many interpretation in $sem(\Sigma)$
- We consider those answers to q that hold for all models in $sem(\Sigma)$ $\rightsquigarrow$ certain answers

Semantics of queries to Σ

Definition

Given an OBDM system Σ and query q posed to Σ, the set of certain answers to q wrt Σ is

$$cert(q, \Sigma) = \bigcap \{ \; q^M \mid M \in sem(\Sigma) \; \}$$

- Query answering in OBDM means to compute the certain answers, i.e., it corresponds to logical implication
- Complexity is usually measured *wrt the size of the source db* $\mathcal{S}$, i.e., we consider data complexity
- When we want to look at query answering as a decision problem, we consider the problem of deciding whether a given tuple $\vec{\mathbf{c}}$ is a certain answer to q wrt Σ, i.e., whether $\vec{\mathbf{c}} \in cert(q, \Sigma)$

Which languages?

- Which language for expressing the ontology?
  - We use Description Logics (OWL), but which one?
- Which language for expressing the mappings?
  - We use logic, but which fragment?
- Which language for expressing queries over the ontology?
  - At least classical conjunctive queries, but we aim at using `SPARQL`

Challenge: optimal compromise between expressive power and data complexity.

Outline

1. Ontology-based data management: The framework
2. Queries in OBDM
3. The nature of query answering in OBDM

Abstracting from the mapping

For the moment, let us abstract from the mapping: we assume that all the semantics of mappings can be captured by computing $\mathcal{M}(\mathcal{S})$, which is the database obtained by treating mappings as assertions translating the data at the sources into facts expressed over the alphabet of the ontology.

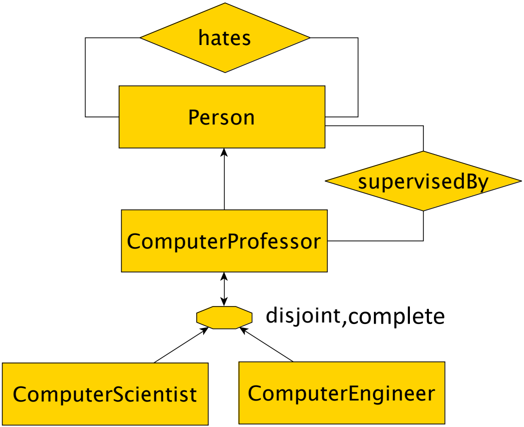
$\mathcal{M}(\mathcal{S})$ can indeed be seen as a set of facts built on the alphabet of $\mathcal{O}$ (i.e., a set of ground atomic formulas in logic, or simply, an ABox, in DL terminology). In other words, formally, we can consider our system as constituted by the pair

$$\langle \mathcal{O}, \mathcal{A} \rangle$$

where $\mathcal{O}$ is the TBox, and $\mathcal{A}$ is the (virtual) ABox.

In practice, instead of computing $\mathcal{M}(\mathcal{S})$ and consider queries over such set of facts, one can use $\mathcal{M}$ to rewrite a query expressed over $\mathcal{M}(\mathcal{S})$ into a query expressed over $\mathcal{S}$, using $\mathcal{M}$.

Which ontology language?



$$q(x) \leftarrow \text{supervisedBy}(x,y), \text{ComputerScientist}(y), \text{hates}(y,z), \text{ComputerEngineering}(z)$$

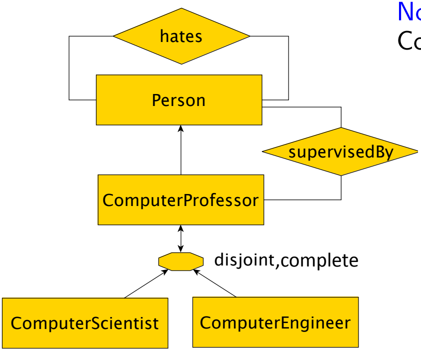
Query answering (QA)

Question

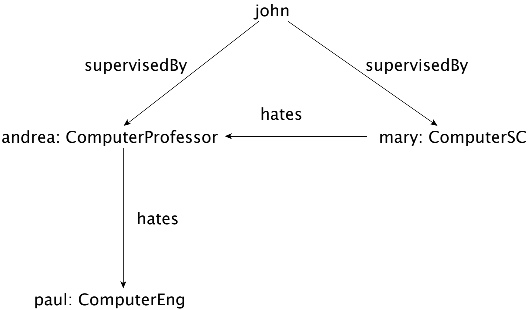
Is ontology-based query answering essentially the same problem as query answering in databases?

In other words, is query answering just evaluating a formula over a (finite) intepretation?

QA in OBDM – Example$^{(*)}$

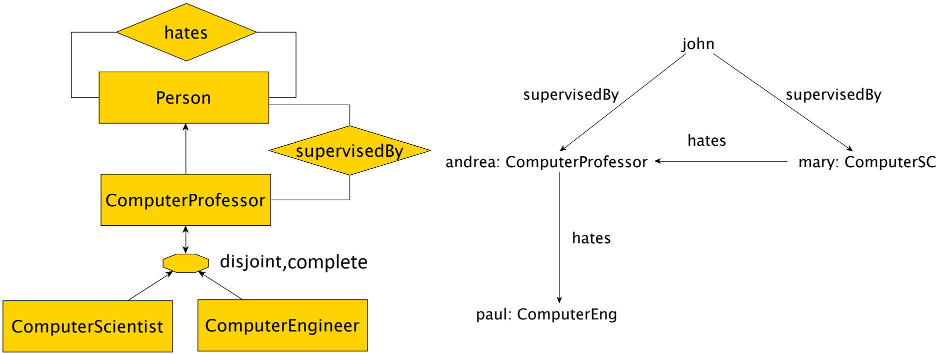


Note that ComputerProfessor is partitioned into ComputerScientist and ComputerEngineer.



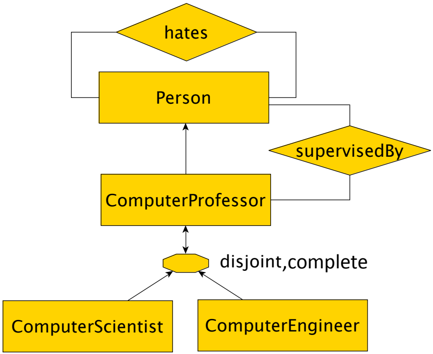
(∗) [Andrea Schaerf 1993]

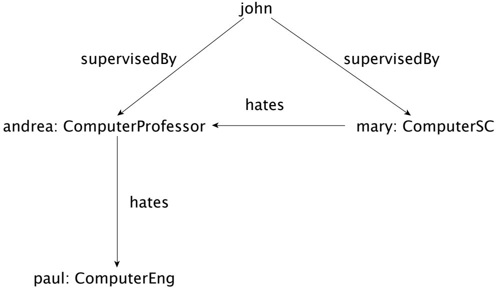
QA in OBDM – Example (cont'd)



$$q(x) \leftarrow \text{supervisedBy}(x, y), \text{ComputerScientist}(y),$$
$$\text{hates}(y, z), \text{ComputerEngineer}(z)$$

Answer: ???





$$q(x) \leftarrow \text{supervisedBy}(x,y), \text{ComputerScientist}(y), \text{hates}(y,z), \text{ComputerEngineer}(z)$$

Answer: { john }

To determine this answer, we need to resort to reasoning by cases on the instances.

Complexity of conjunctive query answering in DLs

	Combined complexity	Data complexity
Plain databases	NP-complete	in LogSpace [(1)]
OWL 2	?	coNP-hard [(2)]

(1) Going beyond probably means not scaling with the data.
(2) Already for a TBox with a single disjunction (see example above).

Questions

- Can we find interesting DLs for which the query answering problem can be solved efficiently (in LogSpace wrt data complexity)?
- If yes, can we leverage relational database technology for query answering in OBDM?

Complexity of conjunctive query answering in DLs

	Combined complexity	Data complexity
Plain databases	NP-complete	in LogSpace [1]
OWL 2	?	coNP-hard [2]

[1] Going beyond probably means not scaling with the data.
[2] Already for a TBox with a single disjunction (see example above).

Questions

- Can we find interesting DLs for which the query answering problem can be solved efficiently (in LogSpace wrt data complexity)?
- If yes, can we leverage relational database technology for query answering in OBDM?

Query rewriting

Query answering can always be thought as done in two phases:

1. Rewriting (wrt the ontology): produce from q and the TBox $\mathcal{O}$ a new query $r_{q,\mathcal{O}}$.
2. Query evaluation: evaluate $r_{q,\mathcal{O}}$ over $\mathcal{M}(\mathcal{S})$ seen as a complete database (and without considering $\mathcal{O}$).

   $\leadsto$ $r_{q,\mathcal{O}}$ is the so-called perfect rewriting of q w.r.t. $\mathcal{O}$ exactly when the query evaluation step produces $cert(q, \langle \mathcal{O}, \mathcal{M}(\mathcal{S}) \rangle)$, for every $\mathcal{S}$.

Note: The "always" holds if we pose no restriction on the language in which to express the rewriting $r_{q,\mathcal{O}}$.

Note: if we have built $\mathcal{M}(\mathcal{S})$, then instead of evaluating $r_{q,\mathcal{O}}$ over $\mathcal{M}(\mathcal{S})$, we rewrite $r_{q,\mathcal{O}}$ wrt $\mathcal{M}$, and then we evaluate the resulting query over $\mathcal{S}$.

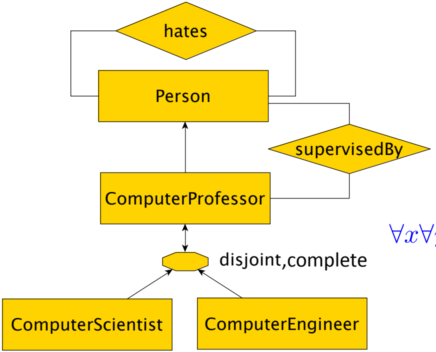
$\mathcal{Q}$-rewritability

Let $\mathcal{Q}$ be a class of queries (or query language) and $\mathcal{L}$ an ontology language.

Def.: $\mathcal{Q}$-rewritability

Query answering is $\mathcal{Q}$-rewritable if for every TBox $\mathcal{O}$ of $\mathcal{L}$ and for every query q, the perfect rewriting $r_{q,\mathcal{O}}$ of q w.r.t. $\mathcal{O}$ can be expressed in the query language $\mathcal{Q}$.

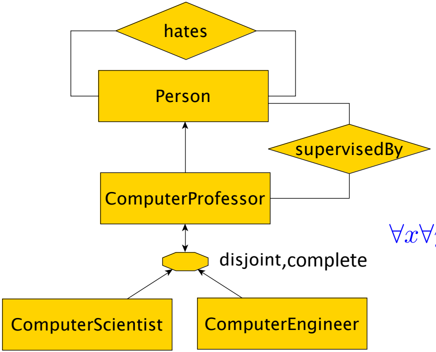
The notion of FOL-rewritability is particularly interesting, where FOL denotes the class of queries expressible in First-Order Logic.



$$\forall x \forall y \ \mathsf{ComputerScientist}(x) \land \mathsf{hates}(x,y) \rightarrow \mathsf{ComputerScientist}(y)$$

$$q(x) \ \leftarrow \ \mathsf{ComputerScientist}(x)$$

QA in OBDM – Example



$\forall x \forall y$ ComputerScientist$(x) \wedge$ hates$(x, y) \rightarrow$ ComputerScientist(y)

$q(x) \leftarrow$ ComputerScientist(x)

The certain answers to the above query are computed by evaluating:

$q'(x) \leftarrow$ ComputerScientist(x)
$q'(x) \leftarrow$ ComputerScientist(y), hates$^+(y, x)$

It can indeed be shown that we need transitive closure in the language of the rewriting.

Complexity of query answering in DLs

Questions

- Can we find interesting DLs for which query answering is FOL-rewritable?
- Even more specifically, can we find interesting DLs for which query answering is UQC-rewritable?

If yes, we can indeed leverage relational database technology for query answering in OBDM (RDBMs are generally very good at optimizing UCQs).

Language of the rewriting

The expressiveness of the ontology language affects the query language into which we are able to rewrite CQs:

- When we can rewrite into UCQ.
  $\rightsquigarrow$ Query evaluation can be "optimized" via RDBMS
- When we can rewrite into FOL/SQL.
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via RDBMS
- When we can rewrite into non recursive Datalog.
  $\rightsquigarrow$ Query evaluation can be still done via RDBMS, but with subqueries/views
- When we can rewrite into an NLogSpace-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\rightsquigarrow$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) Disjunctive Datalog.