Answering Queries in Description Logics: Theory and Applications to Data Management

Diego Calvanese[1], Michael Zakharyaschev[2]

[1] KRDB Research Centre
Free University of Bozen-Bolzano

[2] Birbeck College, London

ESSLLI 2010, August 14–20, 2010
Copenhagen, Denmark

Overview of the Course

1. Introduction and background
    1. Ontology-based data management
    2. Brief introduction to computational complexity
    3. Query answering in databases
    4. Querying databases and ontologies
2. Lightweight description logics
    5. Introduction to description logics
    6. DLs for conceptual data modeling: the *DL-Lite* family
    7. The $\mathcal{EL}$ family of tractable description logics
3. Query answering in the *DL-Lite* family
    8. Query answering in description logics
    9. Lower bounds for more expressive description logics
    10. Query answering by rewriting
4. The combined approach to query answering
    11. Query answering in *DL-Lite*: data completion
    12. Query rewriting in $\mathcal{EL}$
5. Linking ontologies to relational data
    13. The impedance mismatch problem
    14. Query answering in Ontology-Based Data Access systems
6. Conclusions and references

Lecture I

Introduction and background

Outline of Lecture 1

Outline of Lecture 1

1. Ontology-based data management
   - Challenges in data management
   - Description logics and ontologies for data management
   - Requirements on description logics for data management
2. Brief introduction to computational complexity
3. Query answering in databases
4. Querying databases and ontologies
5. References

Outline of Lecture 1

1. Ontology-based data management
   - Challenges in data management
   - Description logics and ontologies for data management
   - Requirements on description logics for data management
2. Brief introduction to computational complexity
3. Query answering in databases
4. Querying databases and ontologies
5. References

New challenges in data management

One of the key challenges in complex systems today is the management of data:

- The **amount** of data has increased enormously.
- The **complexity** of the data has increased:
  structured $\rightsquigarrow$ semi-structured $\rightsquigarrow$ unstructured
- The data may be of **low quality**, e.g., incomplete, inconsistent, not *crisp*.
- Data is increasingly **distributed** and **heterogeneous**, but nevertheless needs to be accessed in a uniform way.
- Data needs to be consumed not only by humans, but also by machines.

Traditional data management systems are not sufficient anymore to fulfill today's data management requirements.

Addressing data management challenges

Several efforts come from the database area:

- New kinds of databases are studied, to manage semi-structured (XML), and probabilistic data.
- Data integration is one of the major challenges for the future or IT.
  E.g., the market for data integration software is estimated to grow from $2.5 billion in 2007 to $3.8 billion in 2012 (+8.7% per year) [IDC. Worldwide Data Integration and Access Software 2008-2012 Forecast. Doc No. 211636 (2008)].

On the other hand, management of complex kinds of information has traditionally been the concern of **Knowledge Representation** in AI:

- Research in AI and KR can bring new insights, solutions, techniques, and technologies.
- **However, what has been done in KR needs to be adapted / extended / tuned to address the new challenges coming from today's data management.**

Outline of Lecture 1

Description logics

Description Logics [Baader *et al.*, 2003] are an important area of KR, studied for the last 25 years, that provide the foundations for the structured representation of information:

- By grounding the used formalisms in logic, the information is provided with a **formal semantics** (i.e., a meaning).
- The logic-based formalization allows one to provide **automated support** for tasks related to data management, by means of **logic-based inference**.
- **Computational aspects** are of concern, so that **tools** can provide **effective support** for automated reasoning.

In this course we are looking into using description logics for data management.

Ontologies

Description logics provide the formal foundations for ontology languages.

Def.: Ontology

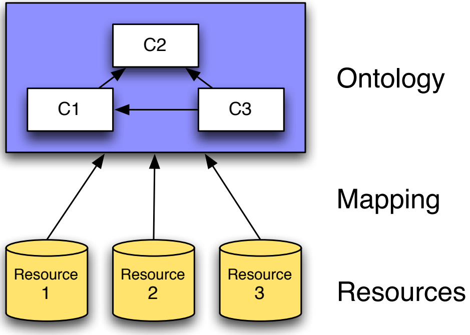
is a representation scheme that describes a **formal conceptualization** of a domain of interest.

The specification of an ontology usually comprises two distinct levels:

- **Intensional level**: specifies a set of conceptual elements and of constraints/axioms describing the conceptual structures of the domain.
- **Extensional level**: specifies a set of instances of the conceptual elements described at the intensional level.

Note: we do not consider here the meta-level, which may also be present in an ontology.

Ontologies at the core of information systems



The usage of all system resources (data and services) is done through the domain conceptualization.

Ontology mediated access to data

Desiderata: achieve **logical transparency** in access to data:

- **Hide** to the user where and how data are stored.
- Present to the user a **conceptual view** of the data.
- Use a **semantically rich formalism** for the conceptual view.

This setting is similar to the one of Data Integration. The difference is that here the ontology provides a rich conceptual description of the data managed by the system.

Issues in ontology-based data access

1. Choice of the formalisms to adopt
2. Efficiency and scalability
3. Tool support

Issue 1: Formalisms to adopt

1. Which is the right ontology language?
   - many proposals have been made
   - differ in expressive power and in complexity of inference
2. Which languages should we use for querying?
   - requirements for querying are different from those for modeling
3. How do we connect the ontology to the data sources?
   - mismatch between information in an ontology and data in the sources

In this course:

- We discuss variants of ontology languages suited for **ontology-based data management**, and study their logical and computational properties.
- We study the problem of querying data through ontologies.
- We briefly discuss problems and solutions related to the impedance mismatch.

Issue 2: Efficiency and scalability

- How can we handle very large ontologies?
  - We have to take into account the **tradeoff** between **expressive power** and **complexity** of inference.
- How can we cope with very large amounts of data?
  - What may be good for large ontologies, may not be good enough for large amounts of data.
- Can we handle multiple data sources and/or multiple ontologies?

In this course:

- We discuss in depth the above mentioned **tradeoff** ...
- ... paying particular attention to the aspects related to **data management**.
- We do not deal with the problem of integrating multiple data sources, since technological issues play a role there.

Issue 3: Tools

- According to the principle that "there is no meaning without a language with a formal semantics", the formal semantics becomes the solid basis for dealing with ontologies.
- Hence every kind of access to an ontology (to extract information, to modify it, etc.), requires to **fully** take into account its semantics.
- We need tools that perform reasoning over the ontology that is **sound and complete** wrt the semantics.
- The tools have to be "efficient", especially wrt the size of the data.

In this course:

- We discuss the requirements, the principles, and the theoretical foundations for ontology-based data access tools.
- We briefly present a tool for querying data sources through ontologies that has been built according to those principles.

Outline of Lecture 1

Which is the "right" expressive power?

What should an ontology language / description logic be able to express in order to be well suited for data management applications?

Let's start with an exercise!

Exercise

Requirements: We are interested in building a software application to manage filmed scenes for realizing a movie, by following the so-called "Hollywood Approach".

Every scene is identified by a code (a string) and is described by a text in natural language.

Every scene is filmed from different positions (at least one), each of this is called a setup. Every setup is characterized by a code (a string) and a text in natural language where the photographic parameters are noted (e.g., aperture, exposure, focal length, filters, etc.). Note that a setup is related to a single scene.

For every setup, several takes may be filmed (at least one). Every take is characterized by a (positive) natural number, a real number representing the number of meters of film that have been used for shooting the take, and the code (a string) of the reel where the film is stored. Note that a take is associated to a single setup.

Scenes are divided into internals that are filmed in a theater, and externals that are filmed in a location and can either be "day scene" or "night scene". Locations are characterized by a code (a string) and the address of the location, and a text describing them in natural language.

Write a precise specification of this domain using any formalism you like!

What do we need to express?

The formalism we use should allow us to express the following:

- domain partitioned into **classes** of objects (e.g., Scene, Location, . . .)
- objects belonging to a class have specific (local) **properties** (e.g., code and text for a scene, . . .)
- **relationships** between objects (e.g., scenes are filmedFrom setups, . . .)
- **inclusions** and **hierarchies** between classes (e.g., Internal and External Scenes)
- **domain** and **range** of relations (e.g., the relation filmedFrom has Scene as domain and Setup as range)
- **mandatory participation** to relations (e.g., every Scene is filmedFrom some Setup)
- **functionality** of relations and attributes (e.g., every Setup is for at most one Scene), and more generally, **numeric constraints**

In addition, we may require:

- **inclusions** and **hierarchies** between relationships
- additional **properties of relationships**, such as transitivity, symmetry, . . .

Solution 1: Use logic!!!

Alphabet: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $filmedFrom(x,y)$, $tkOfStp(x,y)$, $located(x,y)$, $\ldots$.

Axioms:

$\forall x,y.\, code_{Scene}(x,y) \rightarrow Scene(x) \land String(y)$
$\forall x,y.\, description(x,y) \rightarrow Scene(x) \land Text(y)$

$\forall x,y.\, code_{Setup}(x,y) \rightarrow Setup(x) \land String(y)$
$\forall x,y.\, photographicPars(x,y) \rightarrow Setup(x) \land Text(y)$

$\forall x,y.\, nbr(x,y) \rightarrow Take(x) \land Integer(y)$
$\forall x,y.\, filmedMeters(x,y) \rightarrow Take(x) \land Real(y)$
$\forall x,y.\, reel(x,y) \rightarrow Take(x) \land String(y)$

$\forall x,y.\, theater(x,y) \rightarrow Internal(x) \land String(y)$

$\forall x,y.\, nightScene(x,y) \rightarrow External(x) \land Boolean(y)$

$\forall x,y.\, name(x,y) \rightarrow Location(x) \land String(y)$
$\forall x,y.\, address(x,y) \rightarrow Location(x) \land String(y)$
$\forall x,y.\, description(x,y) \rightarrow Location(x) \land Text(y)$

$\forall x.\, Scene(x) \rightarrow (1 \leq \sharp\{y \mid code_{Scene}(x,y)\} \leq 1)$

$\forall x.\, Internal(x) \rightarrow Scene(x)$
$\forall x.\, External(x) \rightarrow Scene(x)$
$\forall x.\, Internal(x) \rightarrow \neg External(x)$
$\forall x.\, Scene(x) \rightarrow Internal(x) \lor External(x)$

$\forall x,y.\, filmedFrom(x,y) \rightarrow Scene(x) \land Setup(y)$
$\forall x,y.\, tkOfStp(x,y) \rightarrow Take(x) \land Setup(y)$
$\forall x,y.\, located(x,y) \rightarrow External(x) \land Location(y)$

$\forall x.\, Scene(x) \rightarrow (1 \leq \sharp\{y \mid filmedFrom(x,y))\}$
$\forall y.\, Setup(y) \rightarrow (1 \leq \sharp\{x \mid filmedFrom(x,y)\} \leq 1)$
$\forall x.\, Take(x) \rightarrow (1 \leq \sharp\{y \mid tkOfStp(x,y)\} \leq 1)$
$\forall x.\, Setup(y) \rightarrow (1 \leq \sharp\{x \mid tkOfStp(x,y)\})$
$\forall x.\, External(x) \rightarrow (1 \leq \sharp\{y \mid located(x,y)\} \leq 1)$
$\ldots$

unibz.it

Solution 1: Use logic – Discussion

Good points:

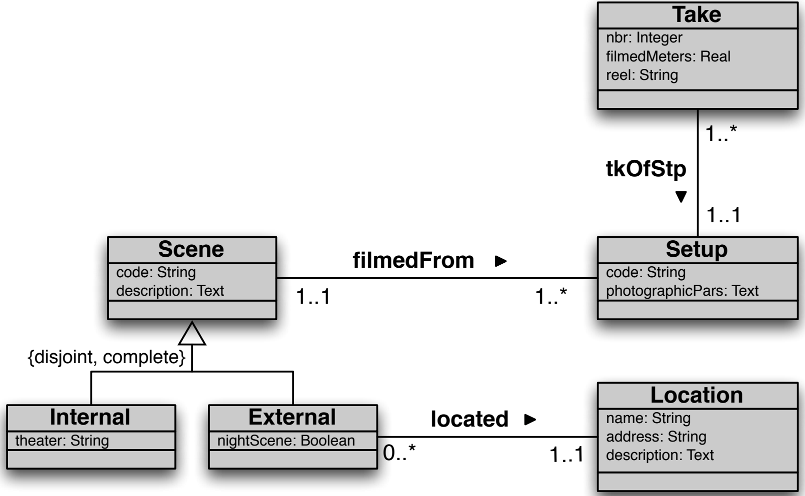
- Precise semantics.
- Formal verification.
- Allows for query answering.
- Machine comprehensible.
- Virtually unlimited expressiveness $^{(*)}$.

Bad points:

- Difficult to generate.
- Difficult to understand for humans.
- Too unstructured (making reasoning difficult), no well-established methodologies available.
- Automated reasoning may be impossible.

$^{(*)}$ *Not really a bad point, in fact.*

Solution 2: Use conceptual modeling diagrams (UML)!



Solution 2: Use conceptual modeling diagrams – Discussion

Good points:

- Easy to generate (it's the standard in software design).
- Easy to understand for humans.
- Well disciplined, well-established methodologies available.

Bad points:

- No precise semantics (people that use it wave hands about it).
- Verification (or better validation) done informally by humans.
- Machine incomprehensible (because of lack of formal semantics).
- Automated reasoning and query answering out of question.
- Limited expressiveness $^{(*)}$.

$^{(*)}$ *Not really a bad point, in fact.*

Solution 3: Use both!!!

Note these two approaches seem to be orthogonal, but in fact they can be used together cooperatively!!!

Basic idea:

- Assign formal semantics to constructs of the conceptual design diagrams.
- Use conceptual design diagrams as usual, taking advantage of methodologies developed for them in Software Engineering.
- Read diagrams as logical theories when needed, i.e., for formal understanding, verification, automated reasoning, etc.

Added values:

- Inherited from conceptual modeling diagrams: ease-to-use for humans
- inherit from logic: formal semantics and reasoning tasks, which are needed for formal verification and machine manipulation.

Solution 3: Use both!!! (cont'd)

Important:

The logical theories that are obtained from conceptual modeling diagrams are of a specific form.

- Their expressiveness is limited (or better, well-disciplined).
- One can exploit the particular form of the logical theory to simplify reasoning.
- The aim is getting:
  - decidability, and
  - reasoning procedures that match the intrinsic computational complexity of reasoning over the conceptual modeling diagrams.

Question

Which are the ontology formalisms / description logics that capture precisely such logical theories?

Outline of Lecture 1

1. Ontology-based data management
2. Brief introduction to computational complexity
   - Basic definitions
   - Hardness and completeness
   - Most important complexity classes
3. Query answering in databases
4. Querying databases and ontologies
5. References

Outline of Lecture 1

Computational complexity (1/2)

[J.E. Hopcroft, 2007; Papadimitriou, 1994]

Computational complexity theory aims at understanding how difficult it is to solve specific problems.

- A **problem** is considered as an (in general infinite) set of instances of the problem, each encoded in some meaningful (i.e., compact) way.
- Standard complexity theory deals with **decision problems**: i.e., problems that admit a yes/no answer.
- **Algorithm** that solves a decision problem:
  - input: an instance of the problem
  - output: yes or no
- The difficulty (complexity) is measured in terms of the amount of **resources** (time, space) that the algorithm needs to solve the problem.
  $\leadsto$ complexity of the algorithm, or **upper bound**
- To measure the complexity of the problem, we consider the best possible algorithm that solves it.
  $\leadsto$ **lower bound**

Computational complexity (2/2)

- **Worst-case** complexity analysis: the complexity is measured in terms of a (complexity) function f:
  - argument: the size n of an instance of the problem (i.e., the length of its encoding)
  - result: the amount $f(n)$ of time/space needed in the worst-case to solve an instance of size n
- The **asymptotic behaviour** of the complexity function when n grows is considered.
- To abstract away from contingent issues (e.g., programming language, processor speed, etc.), we refer to an abstract computing model: **Turing Machines** (TMs).

Complexity classes

To achieve robustness wrt encoding issues, usually one does not consider specific complexity functions f, but rather families $\mathcal{C}$ of complexity functions, giving rise to complexity classes.

Def.: A time/space complexity class $\mathcal{C}$

... is the set of all problems P such that an instance of P of size n can be solved in time/space at most $C(n)$.

Note: Consider a (decision) problem P, and an encoding of the instances of P into strings over some alphabet Σ.
Once we fix such an encoding, the problem actually corresponds to a language L_P, namely the set of strings encoding those instances of the problem for which the answer is yes.

Hence, in the technical sense, a complexity class is actually a set of languages.

Outline of Lecture 1

1. Ontology-based data management
2. Brief introduction to computational complexity
   - Basic definitions
   - Hardness and completeness
   - Most important complexity classes
3. Query answering in databases
4. Querying databases and ontologies
5. References

Reductions

To establish lower bounds on the complexity of problems, we make use of the notion of reduction:

Def.: A **reduction** from a problem P_1 to a problem P_2

... is a function R (the reduction) from instance of P_1 to instances of P_2 such that:

1. R is efficiently computable (i.e., in logarithmic space), and
2. An instance I of P_1 has answer yes iff $R(I)$ has answer yes.

P_1 **reduces to** P_2 if there is a reduction R from P_1 to P_2.

Intuition: If P_1 reduces to P_2, then P_2 is at least as difficult as P_1, since we can solve an instance I of P_1 by reducing it to the instance $R(I)$ of P_2 and then solve $R(I)$.

Hardness and completeness

Def.: A problem P is hard for a complexity class $\mathcal{C}$

. . . if every problem in $\mathcal{C}$ can be reduced to P.

Def.: A problem P is complete for a complexity class $\mathcal{C}$ if

1. it is hard for $\mathcal{C}$, and
2. it belongs to $\mathcal{C}$

Intuitively, a problem that is complete for $\mathcal{C}$ is among the hardest problems in $\mathcal{C}$.

Outline of Lecture 1

Tractability and intractability: PTime and NP

Def.: PTime

Set of problems solvable in polynomial time by a deterministic TM.

- These problems are considered **tractable**, i.e., solvable for large inputs.
- Is a robust class (PTime computations compose).

Def.: NP

Set of problems solvable in polynomial time by a non-deterministic TM.

- These problems are believed **intractable**, i.e., unsolvable for large inputs.
- The best known algorithms actually require exponential time.
- Corresponds to a large class of practical problems, for which the following type of algorithm can be used:
  1. Non-deterministically guess a possible solution of polynomial size.
  2. Check in polynomial time that the guessed solutions is good.

Complexity classes above NP

Def.: PSPACE
Set of problems solvable in polynomial space by a deterministic TM.

- Polynomial space is "not really good", since these problems may require exponential time.
- These problems are considered to be more difficult than NP problems.
- Practical algorithms and heuristics work less well than for NP problems.

Def.: EXPTIME
Set of problems solvable in exponential time by a deterministic TM.

- This is the first provably intractable complexity class.
- These problems are considered to be very difficult.

Complexity classes below PTIME

Def.: LOGSPACE and NLOGSPACE

Set of problems solvable in logarithmic space by a (non-)deterministic TM.

- Note: when measuring the space complexity, the size of the input does not count, and only the working memory (TM tape) is considered.
- Note 2: logarithmic space computations compose (this is not trivial).
- Correspond to reachability in undirected and directed graphs, respectively.

Def.: AC^0

Set of problems solvable in constant time using a polynomial number of processors.

- These problems are solvable efficiently even for very large inputs.
- Corresponds to the complexity of model checking a fixed FO formula when the input is the model only.

Relationship between the complexity classes

The following relationships are known:

$$\mathrm{AC}^0 \subsetneq \textsc{LogSpace} \subseteq \textsc{NLogSpace} \subseteq \textsc{PTime} \subseteq$$
$$\subseteq \textsc{NP} \subseteq \textsc{PSpace} \subseteq \textsc{ExpTime}$$

Moreover, we know that $\textsc{PTime} \subsetneq \textsc{ExpTime}$.

Outline of Lecture 1

Outline of Lecture 1

FOL queries

We assume we are given a relational alphabet Σ, i.e., a set of relation symbols, each with an associated arity.

Def.: A FOL query $\varphi(x_1, \ldots, x_k)$ over Σ (of arity k)

... is a FOL formula over Σ with free variables $x_1, \ldots, x_k$.

Such a query is evaluated w.r.t. a FOL interpretation $\mathcal{I}$, and an assignment α of elements of the domain of $\mathcal{I}$ to $x_1, \ldots, x_k$, i.e., we ask whether:

$$\mathcal{I}, \alpha \models \varphi$$

Given a query $\varphi(x_1, \ldots, x_k)$, we denote with $\langle a_1, \ldots, a_k \rangle$ the assignment that assigns a_i to x_i, for $i \in \{1, \ldots, k\}$.

Note:

- The interpretation $\mathcal{I}$ corresponds to the database over which the query is evaluated.
- The assignments α to the free variables of φ such that $\mathcal{I}, \alpha \models \varphi$ are the **answer to φ over $\mathcal{I}$**, denoted $\varphi^{\mathcal{I}}$.

FOL boolean queries

Def.: A **FOL boolean query** is a FOL query without free variables.

Hence, the answer to a boolean query $\varphi()$ is defined as follows:

$$\varphi()^{\mathcal{I}} = \{() \mid \mathcal{I}, \langle\rangle \models \varphi()\}$$

Such an answer is

- $()$, if $\mathcal{I} \models \varphi$
- $\emptyset$, if $\mathcal{I} \not\models \varphi$.

As an obvious convention we read $()$ as "true" and $\emptyset$ as "false".

Query evaluation

Let us consider:

- a **finite alphabet** Σ, i.e., we have a finite number of predicates and functions, and
- a **finite interpretation** $\mathcal{I}$, i.e., an interpretation (over the finite alphabet) for which $\Delta^{\mathcal{I}}$ is finite.

Then we can consider query evaluation as an algorithmic problem, and study its computational properties.

Note: To study the **computational complexity** of the problem, we need to define a corresponding decision problem.

Query evaluation problem

Definitions

- **Query answering problem**: given a finite interpretation $\mathcal{I}$ and a FOL query $\varphi(x_1, \ldots, x_k)$, compute

$$\varphi^{\mathcal{I}} = \{(a_1, \ldots, a_k) \mid \mathcal{I}, \langle a_1, \ldots, a_k \rangle \models \varphi(x_1, \ldots, x_k)\}$$

- **Recognition problem (for query answering)**: given a finite interpretation $\mathcal{I}$, a FOL query $\varphi(x_1, \ldots, x_k)$, and a tuple $(a_1, \ldots, a_k)$, with $a_i \in \Delta^{\mathcal{I}}$, check whether $(a_1, \ldots, a_k) \in \varphi^{\mathcal{I}}$, i.e., whether

$$\mathcal{I}, \langle a_1, \ldots, a_k \rangle \models \varphi(x_1, \ldots, x_k)$$

Note: The recognition problem for query answering is the decision problem corresponding to the query answering problem.

Query evaluation – Complexity measures [Vardi, 1982]

Def.: Combined complexity

The **combined complexity** is the complexity of $\{\langle\mathcal{I},\alpha,\varphi\rangle \mid \mathcal{I},\alpha \models \varphi\}$, i.e., interpretation, tuple, and query are all considered part of the input.

Def.: Data complexity

The **data complexity** is the complexity of $\{\langle\mathcal{I},\alpha\rangle \mid \mathcal{I},\alpha \models \varphi\}$, i.e., the query φ is fixed (and hence not considered part of the input).

Def.: Query complexity

The **query complexity** is the complexity of $\{\langle\alpha,\varphi\rangle \mid \mathcal{I},\alpha \models \varphi\}$, i.e., the interpretation $\mathcal{I}$ is fixed (and hence not considered part of the input).

Outline of Lecture 1

(Union of) Conjunctive queries – (U)CQs

(Unions of) **conjunctive queries** are an important class of queries:

- A (U)CQ is a FOL query using only conjunction, existential quantification (and disjunction).
- Hence, UCQs contain no negation, no universal quantification, and no function symbols besides constants.
- Correspond to SQL/relational algebra **(union) select-project-join (SPJ) queries** – the most frequently asked queries.
- (U)CQs exhibit nice computational and semantic properties, and have been studied extensively in database theory.
- They are important in practice, since relational database engines are specifically optimized for CQs.

Definition of conjunctive queries

Def.: A **conjunctive query (CQ)** is a FOL query of the form

$$\exists \vec{y}. conj(\vec{x}, \vec{y})$$

where $conj(\vec{x}, \vec{y})$ is a conjunction of atoms and equalities over the free variables $\vec{x}$, the existentially quantified variables $\vec{y}$, and possibly constants.

Note:

- CQs contain no disjunction, no negation, no universal quantification, and no function symbols besides constants.
- Hence, they correspond to relational algebra **select-project-join (SPJ) queries**.
- CQs are the most frequently asked queries.

Conjunctive queries and SQL – Example

Relational alphabet:
Person(name, age), Lives(person, city), Manages(boss, employee)

Query: return name and age of all persons that live in the same city as their boss.

Expressed in SQL:

```
SELECT P.name, P.age
FROM Person P,  Manages M,  Lives L1,  Lives L2
WHERE P.name = L1.person  AND  P.name = M.employee  AND
      M.boss = L2.person  AND  L1.city = L2.city
```

Expressed as a CQ: (the distinguished variables are the blue ones)

$$\exists b, e, p_1, c_1, p_2, c_2.\mathsf{Person}(n, a) \wedge \mathsf{Manages}(b, e) \wedge \mathsf{Lives}(p1, c1) \wedge \mathsf{Lives}(p2, c2) \wedge n = p1 \ \wedge\ n = e \ \wedge\ b = p2 \ \wedge\ c1 = c2$$

Or simpler: $\exists b, c.\mathsf{Person}(n, a) \wedge \mathsf{Manages}(b, n) \wedge \mathsf{Lives}(n, c) \wedge \mathsf{Lives}(b, c)$

Datalog notation for CQs

A CQ $q = \exists\vec{y}.conj(\vec{x},\vec{y})$ can also be written using **datalog notation** as

$$q(\vec{x}_1) \leftarrow conj'(\vec{x}_1,\vec{y}_1)$$

where $conj'(\vec{x}_1,\vec{y}_1)$ is the list of atoms in $conj(\vec{x},\vec{y})$ obtained by equating the variables $\vec{x}$, $\vec{y}$ according to the equalities in $conj(\vec{x},\vec{y})$.

As a result of such an equality elimination, we have that $\vec{x}_1$ and $\vec{y}_1$ can contain constants and multiple occurrences of the same variable.

Def.: In the above query q, we call:

- $q(\vec{x}_1)$ the **head**;
- $conj'(\vec{x}_1,\vec{y}_1)$ the **body**;
- the variables in $\vec{x}_1$ the **distinguished variables**;
- the variables in $\vec{y}_1$ the **non-distinguished variables**.

Conjunctive queries – Example

- Consider the alphabet $\Sigma = \{E/2\}$ and an **interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. Note that $E^{\mathcal{I}}$ is a binary relation, i.e., $\mathcal{I}$ is a directed graph.
- The following **CQ** q returns all nodes that participate to a triangle in the graph:

$$\exists y, z. E(x, y) \wedge E(y, z) \wedge E(z, x)$$

- The query q in **datalog notation** becomes:

$$q(x) \leftarrow E(x, y), E(y, z), E(z, x)$$

- The query q in **SQL** is (we use `Edge(f,s)` for $E(x, y)$):

```
SELECT E1.f
FROM Edge E1, Edge E2, Edge E3
WHERE E1.s = E2.f AND E2.s = E3.f AND E3.s = E1.f
```

Nondeterministic evaluation of CQs

Since a CQ contains only existential quantifications, we can evaluate it by:

1. **guessing a truth assignment** for the non-distinguished variables;
2. **evaluating** the resulting formula (that has no quantifications).

We define a boolean function for CQ evaluation:

```
boolean ConjTruth(𝓘,α,∃y⃗.conj(x⃗,y⃗)) {
  GUESS assignment α[y⃗ ↦ a⃗] {
     return Truth(𝓘,α[y⃗ ↦ a⃗],conj(x⃗,y⃗));
}
```

where $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$ is defined inductively as follows.

Nondeterministic CQ evaluation algorithm

```
boolean Truth(𝓘,α,φ) {
  if (φ is t_1 = t_2)
    return TermEval(𝓘,α,t_1) = TermEval(𝓘,α,t_2);
  if (φ is P(t_1,...,t_k))
    return P^𝓘(TermEval(𝓘,α,t_1),...,TermEval(𝓘,α,t_k));
  if (φ is ψ ∧ ψ')
    return Truth(𝓘,α,ψ) ∧ Truth(𝓘,α,ψ');
}

Δ^𝓘 TermEval(𝓘,α,t) {
   if (t is a variable x) return α(x);
   if (t is a constant c) return c^𝓘;
}
```

CQ evaluation – Combined, data, and query complexity

Theorem (Combined complexity of CQ evaluation)

$\{\langle \mathcal{I}, \alpha, q\rangle \mid \mathcal{I}, \alpha \models q\}$ is NP-**complete** — see below for hardness.

- time: exponential
- space: polynomial

Theorem (Data complexity of CQ evaluation)

$\{\langle \mathcal{I}, \alpha\rangle \mid \mathcal{I}, \alpha \models q\}$ is **in** AC^0

- time: polynomial
- space: logarithmic

Theorem (Query complexity of CQ evaluation)

$\{\langle \alpha, q\rangle \mid \mathcal{I}, \alpha \models q\}$ is NP-**complete** — see below for hardness.

- time: exponential
- space: polynomial

3-colorability

An undirected graph is k-**colorable** if it is possible to assign to each node one of k colors in such a way that every two nodes connected by an edge have different colors.

Def.: 3-colorability is the following decision problem

Given an undirected graph $G = (V, E)$, is it 3-colorable?

Theorem

3-colorability is NP-complete.

We exploit 3-colorability to show NP-hardness of conjunctive query evaluation.

Reduction from 3-colorability to CQ evaluation

Let $G = (V, E)$ be an undirected graph (without edges connecting a node to itself). We consider a relational alphabet consisting of a single binary relation Edge, and we define:

- An **Interpretation**: $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where:
  - $\Delta^{\mathcal{I}} = \{\mathsf{r}, \mathsf{g}, \mathsf{b}\}$
  - $\mathsf{Edge}^{\mathcal{I}} = \{(\mathsf{r}, \mathsf{g}), (\mathsf{g}, \mathsf{r}), (\mathsf{r}, \mathsf{b}), (\mathsf{b}, \mathsf{r}), (\mathsf{g}, \mathsf{b}), (\mathsf{b}, \mathsf{g})\}$
- A **conjunctive query:** Let $V = \{v_1, \ldots, v_n\}$, then consider the boolean conjunctive query defined as:

$$q_G \;=\; \exists x_1, \ldots, x_n. \bigwedge_{\{v_i, v_j\} \in E} \mathsf{Edge}(x_i, x_j) \land \mathsf{Edge}(x_j, x_i)$$

Theorem

G is 3-colorable iff $\mathcal{I} \models q_G$.

NP-hardness of CQ evaluation

The previous reduction immediately gives us the hardness for combined complexity.

Theorem

CQ evaluation is NP-hard in combined complexity.

Note: in the previous reduction, the interpretation does not depend on the actual graph. Hence, the reduction provides also the lower-bound for query complexity.

Theorem

CQ evaluation is NP-hard in query (and combined) complexity.

Outline of Lecture 1

1. Ontology-based data management
2. Brief introduction to computational complexity
3. Query answering in databases
   - First-order logic queries
   - Conjunctive queries
   - Unions of conjunctive queries
4. Querying databases and ontologies
5. References

Union of conjunctive queries (UCQs)

Def.: A **union of conjunctive queries (UCQ)** is a FOL query of the form

$$\bigvee_{i=1,\ldots,n} \exists \vec{y}_i.\mathit{conj}_i(\vec{x}, \vec{y}_i)$$

where each $\exists \vec{y}_i.\mathit{conj}_i(\vec{x}, \vec{y}_i)$ is a conjunctive query (note that all CQs in a UCQ have the same set of distinguished variables).

Note: Obviously, each conjunctive query is also a union of conjunctive queries.

Datalog notation for UCQs

A union of conjunctive queries

$$q = \bigvee_{i=1,\ldots,n} \exists \vec{y_i}.conj_i(\vec{x},\vec{y_i})$$

is written in **datalog notation** as

$$\begin{array}{llll} \{ & q(\vec{x}) & \leftarrow & conj'_1(\vec{x},\vec{y_1}') \\ & & \vdots & \\ & q(\vec{x}) & \leftarrow & conj'_n(\vec{x},\vec{y_n}') \quad \} \end{array}$$

where each element of the set is the datalog expression corresponding to the conjunctive query $q_i = \exists \vec{y_i}.conj_i(\vec{x},\vec{y_i})$.

Note: in general, we omit the set brackets.

Evaluation of UCQs

From the definition of FOL query we have that:

$$\mathcal{I}, \alpha \models \bigvee_{i=1,\ldots,n} \exists \vec{y}_i . conj_i(\vec{x}, \vec{y}_i)$$

if and only if

$$\mathcal{I}, \alpha \models \exists \vec{y}_i . conj_i(\vec{x}, \vec{y}_i) \qquad \text{for some } i \in \{1, \ldots, n\}.$$

Hence to evaluate a UCQ q, we simply evaluate a number (linear in the size of q) of conjunctive queries in isolation.

Hence, **evaluating UCQs has the same complexity as evaluating CQs**.

UCQ evaluation – Combined, data, and query complexity

Theorem (Combined complexity of UCQ evaluation)

$\{\langle \mathcal{I}, \alpha, q\rangle \mid \mathcal{I}, \alpha \models q\}$ is NP-**complete**.

- time: exponential
- space: polynomial

Theorem (Data complexity of UCQ evaluation)

$\{\langle \mathcal{I}, q\rangle \mid \mathcal{I}, \alpha \models q\}$ is **in** AC^0 (query q fixed).

- time: polynomial
- space: logarithmic

Theorem (Query complexity of UCQ evaluation)

$\{\langle \alpha, q\rangle \mid \mathcal{I}, \alpha \models q\}$ is NP-**complete** (interpretation $\mathcal{I}$ fixed).

- time: exponential
- space: polynomial

Outline of Lecture 1

Query answering

In ontology-based data access we are interested in a reasoning service that is not typical in ontologies (or in a FOL theory, or in UML class diagrams, or in a knowledge base) but it is very common in databases: **query answering**.

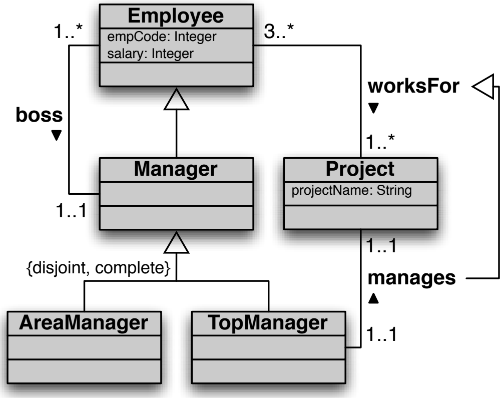
Def.: Query

Is an expression at the intensional level denoting a set of tuples of individuals satisfying a given condition.

Def.: Query Answering

Is the reasoning service that actually computes the answer to a query.

Example of query



$$q(ce, cm, sa) \leftarrow \exists e, p, m. \\ \mathsf{worksFor}(e, p) \wedge \mathsf{manages}(m, p) \wedge \mathsf{boss}(m, e) \wedge \mathsf{empCode}(e, ce) \wedge \\ \mathsf{empCode}(m, cm) \wedge \mathsf{salary}(e, sa) \wedge \mathsf{salary}(m, sa)$$

Query answering under different assumptions

There are two fundamentally different assumptions when addressing query answering:

- **Complete information** on the data, as in traditional databases.
- **Incomplete information** on the data, as in ontologies, but also information integration in databases.

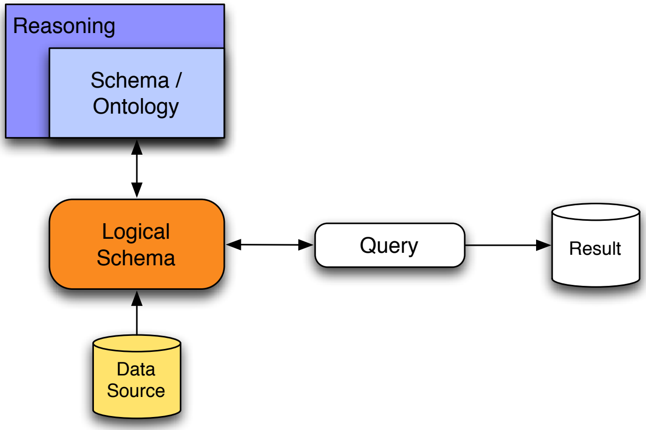
Outline of Lecture 1

Query answering in traditional databases

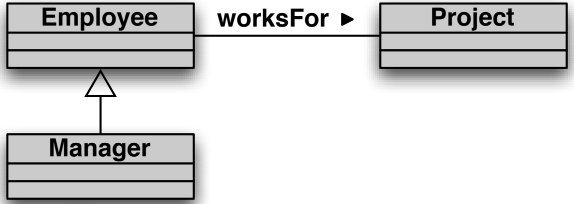
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

$\rightsquigarrow$ Query answering amounts to **query evaluation**, which is computationally easy.

Query answering in traditional databases (cont'd)



Query answering in traditional databases – Example



For each concept/relationship we have a (complete) table in the DB.

DB: Employee = { john, mary, nick }
Manager = { john, nick }
Project = { prA, prB }
worksFor = { (john,prA), (mary,prB) }

Query: $q(x) \leftarrow \exists p.\, \mathsf{Manager}(x) \land \mathsf{Project}(p) \land \mathsf{worksFor}(x,p)$

Answer: { john }

Outline of Lecture 1

1. Ontology-based data management
2. Brief introduction to computational complexity
3. Query answering in databases
4. Querying databases and ontologies
   - Query answering in traditional databases
   - Query answering in ontologies
   - Query answering in ontology-based data access
5. References

Query answering in ontologies

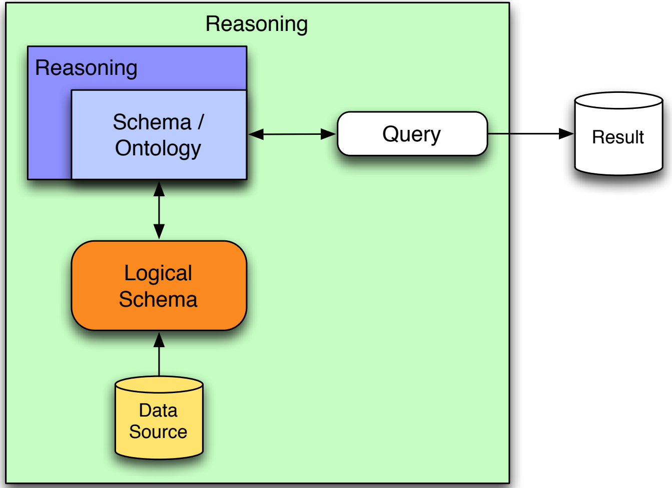
- An ontology (or conceptual schema, or knowledge base) imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.

$\rightsquigarrow$ Query answering amounts to **logical inference**, which is computationally more costly.

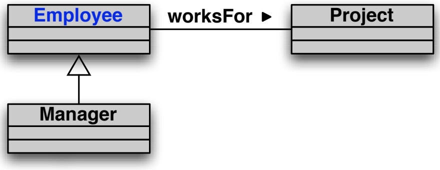
Note:

- The size of the data is not considered critical (comparable to the size of the intensional information).
- Queries are typically simple, i.e., atomic (a class name), and query answering amounts to instance checking.

Query answering in ontologies (cont'd)



Query answering in ontologies – Example



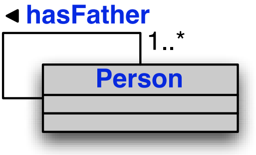
The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

DB: Manager $\supseteq$ { john, nick }
Project $\supseteq$ { prA, prB }
worksFor $\supseteq$ { (john,prA), (mary,prB) }

Query: $q(x) \leftarrow \text{Employee}(x)$

Answer: { john, nick, mary }

Query answering in ontologies – Example 2



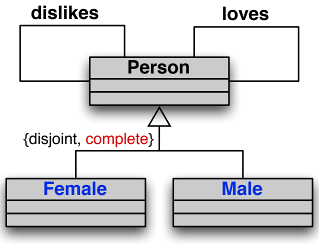
Each person has a father, who is a person.

DB: Person $\supseteq$ { john, nick, toni }
hasFather $\supseteq$ { (john,nick), (nick,toni) }

Queries: $q_1(x, y) \leftarrow$ hasFather(x, y)
$q_2(x) \leftarrow \exists y.\,$hasFather$(x, y)$
$q_3(x) \leftarrow \exists y_1, y_2, y_3.\,$hasFather$(x, y_1) \land$ hasFather$(y_1, y_2) \land$ hasFather(y_2, y_3)
$q_4(x, y_3) \leftarrow \exists y_1, y_2.\,$hasFather$(x, y_1) \land$ hasFather$(y_1, y_2) \land$ hasFather(y_2, y_3)

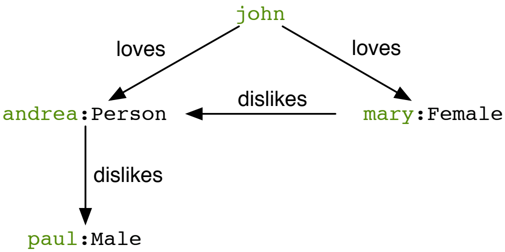
Answers: to q_1: { (john,nick), (nick,toni) }
to q_2: { john, nick, toni }
to q_3: { john, nick, toni }
to q_4: { }

QA in ontologies – Andrea's Example$^{(*)}$



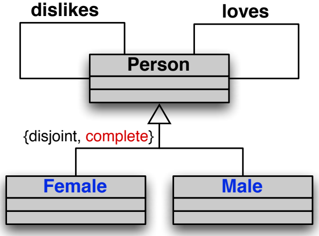
Person is **partitioned into** Female and Male.

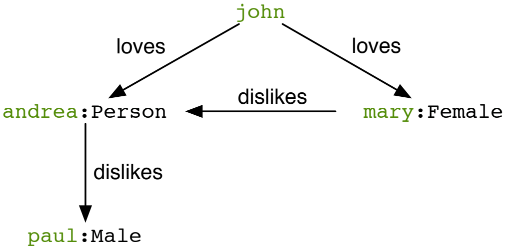
$$\begin{aligned} \text{Person} &\supseteq \{\ \texttt{andrea}, \texttt{paul}, \texttt{mary}, \texttt{john}\ \} \\ \text{Female} &\supseteq \{\ \texttt{mary}\ \} \\ \text{Male} &\supseteq \{\ \texttt{paul}\ \} \\ \text{loves} &\supseteq \{\ (\texttt{john},\texttt{andrea}), (\texttt{john},\texttt{mary})\ \} \\ \text{dislikes} &\supseteq \{\ (\texttt{mary},\texttt{andrea}), (\texttt{andrea},\texttt{paul})\ \} \end{aligned}$$



$^{(*)}$ Due to Andrea Schaerf [Schaerf, 1993].

QA in ontologies – Andrea's Example (cont'd)





$$q(x) \leftarrow \exists y, z.\, \text{loves}(x, y) \wedge \text{Female}(y) \wedge \text{dislikes}(y, z) \wedge \text{Male}(z)$$

Answer: { john }

To determine this answer, we need to resort to **reasoning by cases**.

Outline of Lecture 1

1. Ontology-based data management
2. Brief introduction to computational complexity
3. Query answering in databases
4. Querying databases and ontologies
   - Query answering in traditional databases
   - Query answering in ontologies
   - Query answering in ontology-based data access
5. References

Query answering in ontology-based data access

In OBDA, we have to face the difficulties of both settings:

- The actual **data** is stored in external information sources (i.e., databases), and thus its size is typically **very large**.
- The ontology introduces **incompleteness** of information, and we have to do logical inference, rather than query evaluation.
- We want to take into account at **runtime** the **constraints** expressed in the ontology.
- We want to answer **complex database-like queries**.
- We may have to deal with multiple information sources, and thus face also the problems that are typical of data integration.

Questions that need to be addressed

In the context of ontology-based data access:

1. Which is the "right" **query language**?
2. Which is the "right" **ontology language**?
3. How can we bridge the **semantic mismatch** between the ontology and the data sources?
4. How can **tools for ontology-based data access** take into account these issues?

Which language to use for querying ontologies?

Two borderline cases:

1. Just classes and properties of the ontology $\leadsto$ instance checking
   - Ontology languages are tailored for capturing intensional relationships.
   - They are quite **poor as query languages**:
     Cannot refer to same object via multiple navigation paths in the ontology, i.e., allow only for a limited form of JOIN, namely chaining.
2. Full SQL (or equivalently, first-order logic)
   - Problem: in the presence of incomplete information, query answering becomes **undecidable** (FOL validity).

A good tradeoff is to use (unions of) **conjunctive queries**.

Outline of Lecture 1

References I

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation and Applications.
Cambridge University Press, 2003.

[J.E. Hopcroft, 2007] J.D. Ullman J.E. Hopcroft, R. Motwani.
Introduction to Automata Theory, Languages, and Computation.
Addison Wesley Publ. Co., 3 edition, 2007.

[Papadimitriou, 1994] Christos H. Papadimitriou.
Computational Complexity.
Addison Wesley Publ. Co., 1994.

[Schaerf, 1993] Andrea Schaerf.
On the complexity of the instance checking problem in concept languages with existential quantification.
J. of Intelligent Information Systems, 2:265–278, 1993.

References II

[Vardi, 1982] Moshe Y. Vardi.
The complexity of relational query languages.
In *Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82)*, pages 137–146, 1982.