A Tutorial on Data Integration

Maurizio Lenzerini

Dipartimento di Informatica e Sistemistica Antonio Ruberti,
Sapienza Università di Roma

DEIS'10 - Data Exchange, Integration, and Streaming
November 7-12, 2010, Schloss Dagstuhl,
GI-Dagstuhl Seminar 10452

Structure of the course

1. Introduction to data integration
   - Motivations
   - Logical formalization
   - Mappings
2. Query answering for relational data
   - Approaches to query answering
   - Canonical database
   - Query rewriting
   - Counterexamples
   - Query containment
3. Beyond relational data
   - Semi-structured data integration
   - Ontology-based data integration

Structure of the course

1. Introduction to data integration
   - Motivations
   - Logical formalization
   - Mappings
2. Query answering for relational data
   - Approaches to query answering
   - Canonical database
   - Query rewriting
   - Counterexamples
   - Query containment
3. Beyond relational data
   - Semi-structured data integration
   - Ontology-based data integration

Structure of the course

1. Introduction to data integration
    - Motivations
    - Logical formalization
    - Mappings

2. Query answering for relational data
    - Approaches to query answering
    - Canonical database
    - Query rewriting
    - Counterexamples
    - Query containment

3. Beyond relational data
    - Semi-structured data integration
    - Ontology-based data integration

Part I

Introduction to data integration

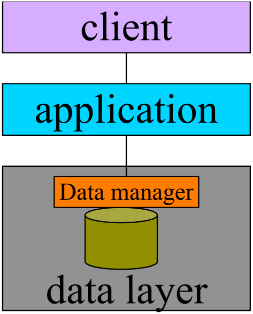
Outline

Outline

1. Motivations
   - What is data integration?
   - Variants of data integration
   - Issues in data integration
2. Data integration: Logical formalization
   - Syntax and semantics of a data integration system
   - Queries to a data integration system
3. Mappings
   - Types of mappings
   - GAV mappings
   - LAV mappings
   - GLAV mappings

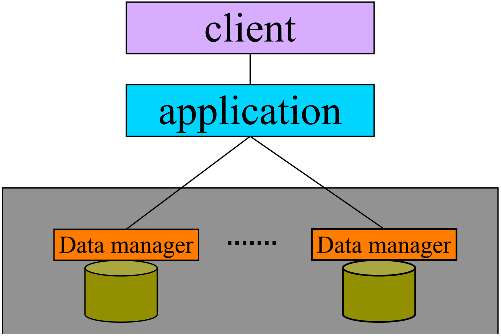
Outline

Integration in data management: evolution



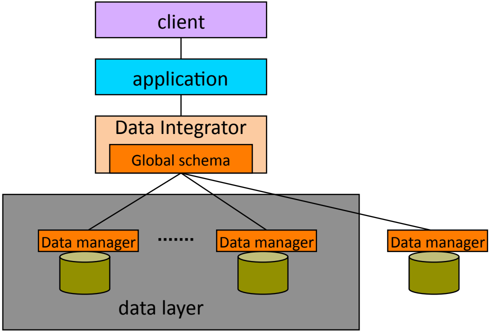
- Centralized system with three-tier architecture
- "Implicit" integration: integration supported by the Data Base Management System (DBMS), i.e., the data manager

Integration in data management: evolution



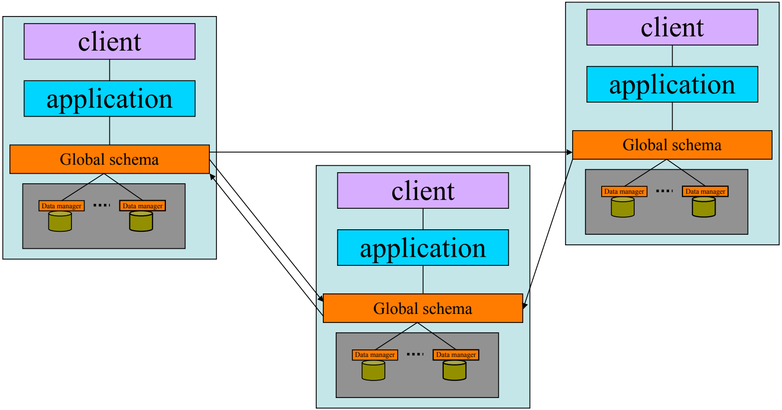
- Centralized system with three-tier architecture and multiple stores
- Application-hidden integration: integration "embedded" within application

Integration in data management: evolution



- Centralized system with four-tier architecture and multiple, distributed stores
- (Centralized) data integration: the global schema is mapped to the different data sources, which are heterogeneous, distributed and autonomous

Integration in data management: evolution



- Decentralized system
- Peer-to-peer data integration: distributed data integration realized with no unique, central global schema

Outline

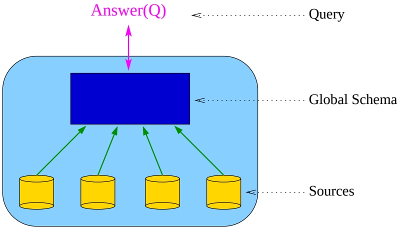
Approaches to data integration

- Centralized, virtual data integration . . . *is the main topic of this tutorial*
- Data warehousing . . . *not dealt with in this tutorial*
- P2P data integration . . . *not dealt with in this tutorial*

Centralized data integration

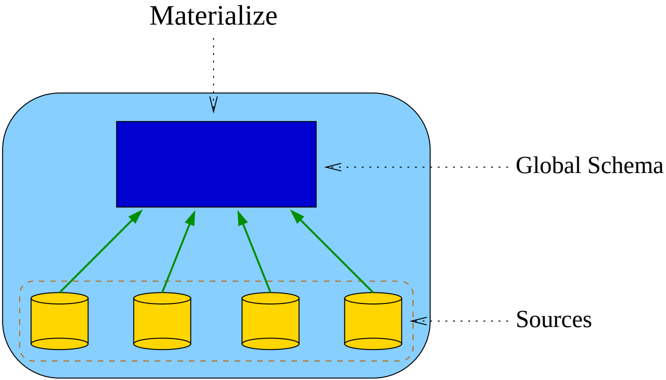
Centralized data integration is the problem of providing unified and transparent view to a collection of data stored in multiple, autonomous, and heterogeneous data sources.

The unified view is achieved through a global (or target) schema, linked to the data sources by means of mappings.

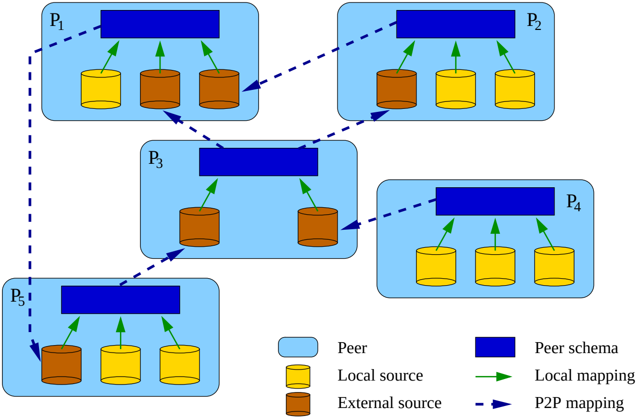


Data warehousing

- materialization of the global database
- allows for OLAP without accessing the sources
- similar to data exchange



Peer-to-peer data integration



Talk 10 – Armin Roth, “Peer data management systems”
Talk 11 – Sebastian Skritek, “Theory of Peer Data Management”

Outline

1. Motivations
   - What is data integration?
   - Variants of data integration
   - Issues in data integration
2. Data integration: Logical formalization
   - Syntax and semantics of a data integration system
   - Queries to a data integration system
3. Mappings
   - Types of mappings
   - GAV mappings
   - LAV mappings
   - GLAV mappings

Main issues in data integration

1. Data extraction, cleaning, and reconciliation
   Talk 9 – Ekaterini Ioannou, "Data cleaning for data integration"
2. How to discover and specify the mappings between sources and global schema
   Talk 22 – Marie Jacob, "Learning and discovering queries and mappings"
3. How to model and specify the global schema
4. How to answer queries expressed on the global schema
   Talk 2 – Piotr Wieczorek, "Query answering in data integration"
5. How to deal with limitations in mechanisms for accessing sources
6. How to optimize query answering
7. . . .

Outline

Outline

Formal framework for data integration

Definition

A data integration system $\mathcal{I}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{G}$ is the global schema
  a logical theory over an alphabet $\mathcal{A}_{\mathcal{G}}$
- $\mathcal{S}$ is the source schema
  an alphabet $\mathcal{A}_{\mathcal{S}}$ *disjoint from* $\mathcal{A}_{\mathcal{G}}$
- $\mathcal{M}$ is the mapping between $\mathcal{S}$ and $\mathcal{G}$
  We consider different approaches to the specification of mappings

Semantics of a data integration system

Which are the dbs that satisfy $\mathcal{I}$, i.e., the logical models of $\mathcal{I}$?

- We refer only to dbs over a fixed infinite domain Δ of elements
- We start from the data present in the sources: these are modeled through a (finite) source database $\mathcal{C}$ over Δ (also called source model), fixing the extension of the predicates of $\mathcal{A}_\mathcal{S}$
- The dbs for $\mathcal{I}$ are logical interpretations for $\mathcal{A}_\mathcal{G}$, called global dbs

Definition

The semantics of $\mathcal{I}$ relative to $\mathcal{C}$ is:

$$sem^{\mathcal{C}}(\mathcal{I}) = \{ \mathcal{B} \mid \mathcal{B} \text{ is a global database that satisfies } \mathcal{G} \text{ and that satisfies } \mathcal{M} \text{ wrt } \mathcal{C} \}$$

To satisfy $\mathcal{G}$ means to satisfy all axioms of $\mathcal{G}$, i.e., being a model of $\mathcal{G}$
What it means to satisfy $\mathcal{M}$ wrt $\mathcal{C}$ depends on the nature of $\mathcal{M}$

Semantics of a data integration system

Which are the dbs that satisfy $\mathcal{I}$, i.e., the logical models of $\mathcal{I}$?

- We refer only to dbs over a fixed infinite domain Δ of elements
- We start from the data present in the sources: these are modeled through a (finite) source database $\mathcal{C}$ over Δ (also called source model), fixing the extension of the predicates of $\mathcal{A}_{\mathcal{S}}$
- The dbs for $\mathcal{I}$ are logical interpretations for $\mathcal{A}_{\mathcal{G}}$, called global dbs

Definition

The semantics of $\mathcal{I}$ relative to $\mathcal{C}$ is:

$sem^{\mathcal{C}}(\mathcal{I}) = \{ \mathcal{B} \mid \mathcal{B}$ is a global database that satisfies $\mathcal{G}$ and that satisfies $\mathcal{M}$ wrt $\mathcal{C}$ $\}$

To satisfy $\mathcal{G}$ means to satisfy all axioms of $\mathcal{G}$, i.e., being a model of $\mathcal{G}$
What it means to satisfy $\mathcal{M}$ wrt $\mathcal{C}$ depends on the nature of $\mathcal{M}$

Comparison between data integration and data exchange

- Data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$
- Data exchange setting $M = \langle \mathcal{S}, \mathcal{T}, \Sigma \rangle$

$\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$	$M = \langle \mathcal{S}, \mathcal{T}, \Sigma \rangle$
$\mathcal{S}$	$\mathcal{S}$
$\mathcal{G}$	$\mathcal{T}$
$\mathcal{M}$	Σ
finite source database $\mathcal{C}$	finite source instance I
global database with no variable	target instance is finite and may contain variables
global database satisfying $\mathcal{G}$ and $\mathcal{M}$ wrt $\mathcal{C}$	solution J

Outline

1. Motivations
   - What is data integration?
   - Variants of data integration
   - Issues in data integration
2. Data integration: Logical formalization
   - Syntax and semantics of a data integration system
   - Queries to a data integration system
3. Mappings
   - Types of mappings
   - GAV mappings
   - LAV mappings
   - GLAV mappings

Queries to a data integration system $\mathcal{I}$

- The domain Δ is fixed, and we do not distinguish an element of Δ from the constant denoting it $\leadsto$ standard names
- Queries to $\mathcal{I}$ are expressions (of a certain arity) over the alphabet $\mathcal{A}_{\mathcal{G}}$; the evaluation of a query of arity n to $\mathcal{I}$ relative to a source database $\mathcal{C}$ returns a set of tuples of elements Δ, each of arity n
- When "evaluating" q over $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we have to consider that for a given source database $\mathcal{C}$, there may be many global databases $\mathcal{B}$ satisfying $\mathcal{G}$ and $\mathcal{M}$ wrt $\mathcal{C}$, i.e., many global databases $\mathcal{B}$ in $sem^{\mathcal{C}}(\mathcal{I})$
- We consider those answers to q that hold for all global databases in $sem^{\mathcal{C}}(\mathcal{I})$

  $\leadsto$ certain answers

Semantics of queries to $\mathcal{I}$

Definition

Given q, $\mathcal{I}$, and $\mathcal{C}$, the set of certain answers to q wrt $\mathcal{I}$ and $\mathcal{C}$ is

$$cert(q,\mathcal{I},\mathcal{C}) = \bigcap \{ \; q^{\mathcal{B}} \mid \forall \; \mathcal{B} \in sem^{\mathcal{C}}(\mathcal{I}) \; \}$$

- Query answering in information integration means to compute the certain answers, i.e., it corresponds to logical implication
- Complexity is measured mainly *wrt the size of the source db* $\mathcal{C}$, 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 $\vec{c}$ is a certain answer to q wrt $\mathcal{I}$ and $\mathcal{C}$, i.e., whether $\vec{c} \in cert(q,\mathcal{I},\mathcal{C})$

Databases with incomplete information, or knowledge bases

- Traditional database: one model of a first-order theory.
  Query answering means evaluating a formula in the model
- Database with incomplete information, or knowledge base: set of models (specified, for example, as a restricted first-order theory).
  Query answering means computing the tuples that satisfy the query in all the models in the set

There is a strong connection between query answering in information integration and query answering in databases with incomplete information under constraints (or, query answering in knowledge bases)

Databases with incomplete information, or knowledge bases

- Traditional database: one model of a first-order theory. Query answering means evaluating a formula in the model
- Database with incomplete information, or knowledge base: set of models (specified, for example, as a restricted first-order theory). Query answering means computing the tuples that satisfy the query in all the models in the set

There is a strong connection between query answering in information integration and query answering in databases with incomplete information under constraints (or, query answering in knowledge bases)

Query answering: problem space

- Global schema
  - Relational data
    - without constraints (i.e., empty theory)
    - with constraints
  - Non-relational data
    - Graph-databases
      Talk 18 – Paolo Guagliardo "View-based query processing"
    - XML-data
      Talk 14 – Lucja Kot, "XML data integration"
    - Ontologies
      Talk 8 – Yazmin A. Ibanez, "Description logics for data integration"
- Mapping
  - GAV, LAV, or GLAV
- Semantics
  - arbitrary vs. finite databases
  - Standard logic vs. Inconsistency-tolerant semantics
    Talk 7 – Slawomir Staworko, "Consistent query answering"

Outline

1. Motivations
   - What is data integration?
   - Variants of data integration
   - Issues in data integration
2. Data integration: Logical formalization
   - Syntax and semantics of a data integration system
   - Queries to a data integration system
3. Mappings
   - Types of mappings
   - GAV mappings
   - LAV mappings
   - GLAV mappings

Outline

The mapping

In this tutorial, we mainly consider sound mappings, i.e., mapping assertions stating that the presence of certain data in the sources implies the presence of certain data in the virtual global database.

How is the mapping $\mathcal{M}$ between $\mathcal{S}$ and $\mathcal{G}$ specified?

- Are the sources defined in terms of the global schema?
  Approach called source-centric, or local-as-view, or LAV
- Is the global schema defined in terms of the sources?
  Approach called global-schema-centric, or global-as-view, or GAV
- A mixed approach?
  Approach called GLAV

GAV vs. LAV – Example

Global schema:

movie($Title$, $Year$, $Director$)
european($Director$)
review($Title$, $Critique$)

Source 1:

r_1($Title$, $Year$, $Director$) since 1960, european directors

Source 2:

r_2($Title$, $Critique$) since 1990

Query: Title and critique of movies in 1998

$\{ (t, r) \mid \exists d.\ \mathsf{movie}(t, 1998, d) \wedge \mathsf{review}(t, r) \}$, abbreviated
$\{ (t, r) \mid \mathsf{movie}(t, 1998, d),\ \mathsf{review}(t, r) \}$

Outline

Formalization of GAV

In GAV (with sound sources), the mapping $\mathcal{M}$ is a set of assertions:

$$\forall \vec{x}.\ \phi_{\mathcal{S}}(\vec{x}) \rightarrow g(\vec{x})$$

one for each element g in $\mathcal{A}_{\mathcal{G}}$, with $\phi_{\mathcal{S}}$ a query over $\mathcal{S}$ of the arity of g

Given a source db $\mathcal{C}$, a db $\mathcal{B}$ for $\mathcal{G}$ satisfies $\mathcal{M}$ wrt $\mathcal{C}$ if for each $g \in \mathcal{G}$:

$$\phi_{\mathcal{S}}^{\mathcal{C}} \subseteq g^{\mathcal{B}}$$

Given a source database $\mathcal{C}$, $\mathcal{M}$ provides direct information about which data in $\mathcal{C}$ satisfy the elements of the global schema

Elements in the global schema $\mathcal{G}$ can be considered as views over the sources. This is why this approach is called "global as view"

GAV – Example

Global schema: movie($Title, Year, Director$)
european($Director$)
review($Title, Critique$)

GAV: to each relation in the global schema, $\mathcal{M}$ associates a view over the sources:

$$\forall t, y, d\ \mathsf{r}_1(t, y, d) \rightarrow \mathsf{movie}(t, y, d)$$
$$\forall d, t, y\ \mathsf{r}_1(t, y, d) \rightarrow \mathsf{european}(d)$$
$$\forall t, r\ \mathsf{r}_2(t, r) \rightarrow \mathsf{review}(t, r)$$

GAV – Example of query processing

The query

$$\{ \ (t, r) \ \mid \ \textsf{movie}(t, 1998, d), \ \textsf{review}(t, r) \ \}$$

is processed by expanding each atom according to its associated definition in $\mathcal{M}$, so as to come up with a query over the source relations

In particular:

$$\{ \ (t, r) \ \mid \ \textsf{movie}(t, 1998, d), \ \ \textsf{review}(t, r) \ \}$$

$$\downarrow \qquad\qquad\qquad \downarrow$$

$$\{ \ (t, r) \ \mid \ \textsf{r}_1(t, 1998, d), \ \ \textsf{r}_2(t, r) \ \}$$

GAV – Example of constraints

Global schema containing constraints:

movie($Title$, $Year$, $Director$)
european($Director$)
review($Title$, $Critique$)
$\forall x, c$ review$(x, c) \rightarrow \exists y, d$ movie(x, y, d)

GAV mappings:

$\forall t, y, d$ $r_1(t, y, d) \rightarrow$ movie(t, y, d)
$\forall d, t, y$ $r_1(t, y, d) \rightarrow$ european(d)
$\forall t, r$ $r_2(t, r) \rightarrow$ review(t, r)

Outline

Formalization of LAV

In LAV (with sound sources), the mapping $\mathcal{M}$ is a set of assertions:

$$\forall \vec{x}.\ s(\vec{x}) \rightarrow \phi_{\mathcal{G}}(\vec{x})$$

one for each source element s in $\mathcal{A}_{\mathcal{S}}$, with $\phi_{\mathcal{G}}$ a query over $\mathcal{G}$ of the arity of s.

Given source db $\mathcal{C}$, a db $\mathcal{B}$ for $\mathcal{G}$ satisfies $\mathcal{M}$ wrt $\mathcal{C}$ if for each $s \in \mathcal{S}$:

$$s^{\mathcal{C}} \subseteq \phi_{\mathcal{G}}^{\mathcal{B}}$$

The mapping $\mathcal{M}$ and the source database $\mathcal{C}$ do not provide direct information about which data satisfy the global schema

Sources, i.e., elements in $\mathcal{S}$, can be considered as views over the global schema. This is why this approach is called "local-as-views".

LAV – Example

Global schema: movie($Title, Year, Director$)
european($Director$)
review($Title, Critique$)

LAV: to each source relation, $\mathcal{M}$ associates a view over the global schema:

$$\mathsf{r}_1(t, y, d) \rightarrow \{ (t, y, d) \mid \mathsf{movie}(t, y, d),\ \mathsf{european}(d),\ y \geq 1960 \}$$
$$\mathsf{r}_2(t, r) \rightarrow \{ (t, r) \mid \mathsf{movie}(t, y, d),\ \mathsf{review}(t, r),\ y \geq 1990 \}$$

The query $\{ (t, r) \mid \mathsf{movie}(t, 1998, d),\ \mathsf{review}(t, r) \}$ is processed by means of an inference mechanism that aims at re-expressing the atoms of the global schema in terms of atoms at the sources.
In this case:

$$\{ (t, r) \mid \mathsf{r}_2(t, r),\ \mathsf{r}_1(t, 1998, d) \}$$

GAV and LAV – Comparison

GAV: (e.g., Carnot, SIMS, Tsimmis, IBIS, Momis, DisAtDis, . . .)

- Quality depends on how well we have compiled the sources into the global schema through the mapping
- Whenever a source changes or a new one is added, the global schema needs to be reconsidered
- Query processing can be based on some sort of unfolding (query answering looks easier – without constraints)

LAV: (e.g., Information Manifold, DWQ, Picsel)

- Quality depends on how well we have characterized the sources
- High modularity and extensibility (if the global schema is well designed, when a source changes, only its definition is affected)
- Query processing needs reasoning (query answering complex)

Outline

Beyond GAV and LAV: GLAV

In GLAV (with sound sources), the mapping $\mathcal{M}$ is a set of assertions:

$$\forall \vec{x}.\ \phi_{\mathcal{S}}(\vec{x}) \rightarrow \phi_{\mathcal{G}}(\vec{x})$$

with $\phi_{\mathcal{S}}$ a query over $\mathcal{S}$, and $\phi_{\mathcal{G}}$ a query over $\mathcal{G}$ of the same arity as $\phi_{\mathcal{S}}$

Given source db $\mathcal{C}$, a db $\mathcal{B}$ for $\mathcal{G}$ satisfies $\mathcal{M}$ wrt $\mathcal{C}$ if for each $\forall \vec{x}.\ \phi_{\mathcal{S}}(\vec{x}) \rightarrow \phi_{\mathcal{G}}(\vec{x})$ in $\mathcal{M}$:

$$\phi_{\mathcal{S}}^{\mathcal{C}} \subseteq \phi_{\mathcal{G}}^{\mathcal{B}}$$

As for LAV, the mapping $\mathcal{M}$ does not provide direct information about which data satisfy the global schema, and, therefore, to answer a query q over $\mathcal{G}$, we have to infer how to use $\mathcal{M}$ in order to access the source database $\mathcal{C}$

GLAV – Example

Global schema: $\text{work}(Person, Project)$, $\text{area}(Project, Field)$

Source 1: $\text{hasjob}(Person, Field)$
Source 2: $\text{teaches}(Professor, Course)$, $\text{in}(Course, Field)$
Source 3: $\text{get}(Researcher, Grant)$, $\text{for}(Grant, Project)$

GLAV mapping:

$$\{(r,f) \mid \text{hasjob}(r,f)\} \rightarrow \{(r,f) \mid \text{work}(r,p),\ \text{area}(p,f)\}$$
$$\{(r,f) \mid \text{teaches}(r,c),\ \text{in}(c,f)\} \rightarrow \{(r,f) \mid \text{work}(r,p),\ \text{area}(p,f)\}$$
$$\{(r,p) \mid \text{get}(r,g),\ \text{for}(g,p)\} \rightarrow \{(r,f) \mid \text{work}(r,p)\}$$

Exact mappings

Although we consider only sound mappings in this tutorial, exact mappings have also been studied in data integration.

An exact GLAV mapping assertion have the form:

$$\forall \vec{x}.\ \phi_{\mathcal{S}}(\vec{x}) \leftrightarrow \phi_{\mathcal{G}}(\vec{x})$$

with $\phi_{\mathcal{S}}$ a query over $\mathcal{S}$, and $\phi_{\mathcal{G}}$ a query over $\mathcal{G}$ of the same arity as $\phi_{\mathcal{S}}$

Given source db $\mathcal{C}$, a db $\mathcal{B}$ for $\mathcal{G}$ satisfies the exact mapping assertion $\forall \vec{x}.\ \phi_{\mathcal{S}}(\vec{x}) \leftrightarrow \phi_{\mathcal{G}}(\vec{x})$ if

$$\phi_{\mathcal{S}}^{\mathcal{C}} = \phi_{\mathcal{G}}^{\mathcal{B}}$$

GAV and LAV exact mapping assertions are defined in the obvious way

Part II

Query answering for relational data

Outline

Outline

Query answering in different settings

The problem of query answering comes in different forms, depending on

- Global schema
  - relational
    - without constraints (i.e., empty theory)
    - with constraints
  - non-relational data
- Mapping
  - GAV
  - LAV (or GLAV)
- Queries
  - user queries
  - queries in the mapping

If not otherwise stated, we will assume that both the user queries and the queries in the mappings are conjunctive queries

Incompleteness and inconsistency

Query answering heavily depends upon whether incompleteness/inconsistency shows up

Incompleteness: the cardinality of $sem^{\mathcal{C}}(\mathcal{I})$ is greater than 1
Inconsistency: the cardinality of $sem^{\mathcal{C}}(\mathcal{I})$ is 0

Constraints in $\mathcal{G}$	Type of mapping	Incompleteness	Inconsistency
no	GAV	very limited	no
no	(G)LAV	yes	no
yes	GAV	yes	yes
yes	(G)LAV	yes	yes

Main approaches to query answering

- Based on canonical database
- Based on query rewriting
- Based on counterexample
- Based on query containment

Outline

Outline

The canonical database

Given data integration system $\mathcal{I}$, and source database $\mathcal{C}$, a canonical database (or, canonical model) for $\mathcal{I}$ and $\mathcal{C}$ is global database $\mathcal{B} \in sem^{\mathcal{C}}(\mathcal{I})$, possibly with variables, such that for each query q on $\mathcal{A}_{\mathcal{G}}$, and each tuple $\vec{t}$, $\vec{t} \in cert(q, \mathcal{I}, \mathcal{C})$ if and only if $\vec{t} \in q^{\mathcal{B}}$ (or, $\vec{t} \in q_1^{\mathcal{B}}$ for a suitable query q_1)

Note the similarity with the notion of universal solution in data exchange

In what follows, we discuss the approach based on canonical database by referring to GAV without constraints, and by limiting the attention to positive user queries

Outline

GAV without constraints – Retrieved global database

Definition

Given a GAV data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a source database $\mathcal{C}$ for $\mathcal{S}$, we call retrieved global database (for $\mathcal{I}$ wrt $\mathcal{C}$), denoted $\mathcal{M}(\mathcal{C})$, the global database obtained by "applying" the queries in the mapping, and "transferring" to the elements of $\mathcal{G}$ the corresponding tuples retrieved from $\mathcal{C}$

Note that, since mappings are of type GAV, the tuples to be "tranferred" to the global schema are definite (they do not contain existentially quantified elements)

GAV without constraints – Example

Consider $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, with

Global schema $\mathcal{G}$: student$(Code, Name, City)$
university$(Code, Name)$
enrolled$(Scode, Ucode)$

Source schema $\mathcal{S}$: relations $s_1(Scode, Sname, City, Age)$,
$s_2(Ucode, Uname)$, $s_3(Scode, Ucode)$

Mapping $\mathcal{M}$:

$$\forall c, n, ci \; s_1(c, n, ci, a) \rightarrow \text{student}(c, n, ci)$$
$$\forall c, n \; s_2(c, n) \rightarrow \text{university}(c, n)$$
$$\forall s, u \; s_3(s, u) \rightarrow \text{enrolled}(s, u)$$

Example of retrieved global database

university

Code	*Name*
AF	bocconi
BN	ucla

student

Code	*Name*	*City*
12	anne	florence
15	bill	oslo

enrolled

Scode	*Ucode*
12	AF
16	BN

$s_1^{\mathcal{C}}$

12	anne	florence	21
15	bill	oslo	24

$s_2^{\mathcal{C}}$

AF	bocconi
BN	ucla

$s_3^{\mathcal{C}}$

12	AF
16	BN

Example of source database $\mathcal{C}$ and corresponding retrieved global database $\mathcal{M}(\mathcal{C})$

GAV without constraints – Canonical database

GAV mapping assertions have the form $\forall \vec{x}.\ \phi_S(\vec{x}) \rightarrow g(\vec{x})$ where ϕ_S is a query over the source relations, and g is an element of $\mathcal{G}$

In general, given a source database $\mathcal{C}$, there are several databases in $sem^{\mathcal{C}}(\mathcal{I})$

However, it is easy to see that, when $\mathcal{G}$ has no axiom, $\mathcal{M}(\mathcal{C})$ is the intersection of all such databases, and therefore, is finite, and is the only "minimal" model of $\mathcal{I}$

For positive queries, $\mathcal{M}(\mathcal{C})$ is a canonical database of $\mathcal{I}$ wrt $\mathcal{C}$: If q is a positive query, then $\vec{t} \in cert(q, \mathcal{I}, \mathcal{C})$ iff $\vec{t} \in q^{\mathcal{M}(\mathcal{C})}$

Exercise 1

Is the following problem decidable?

Given a GAV data integration system $\mathcal{I}$ without constraints, a source database $\mathcal{C}$, a first order logic query q over $\mathcal{A}_{\mathcal{G}}$, compute the certain answers $cert(q, \mathcal{I}, \mathcal{C})$

Extensions to other cases

- (G)LAV without constraints
  the chase constructs a universal solution (with variables)
- GAV and (G)LAV with constraints
  a finite universal solution may not exist

Outline

Outline

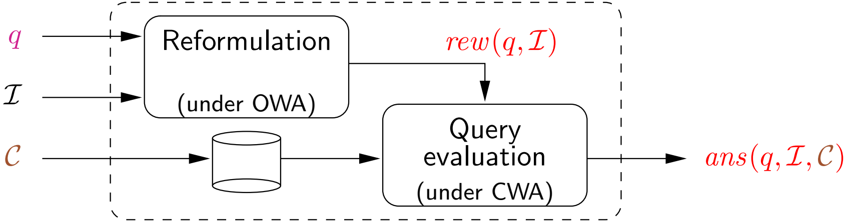
Query answering based on query rewriting

Given data integration system $\mathcal{I}$, and a user query q, compute a query q_1 over $\mathcal{A}_S$, and then compute $q_1^{\mathcal{C}}$

Thus, query answering is divided in two steps:

1. Reformulate the user query in terms of a new query over the alphabet of $\mathcal{A}_S$, called source rewriting, or simply rewriting expressed in a given query language
2. Evaluate the rewriting over the source database $\mathcal{C}$

Query rewriting



The language of $rew(q, \mathcal{I})$ is chosen a priori!

What is a rewriting?

Definition

A query q_1 over the alphabet $\mathcal{A}_\mathcal{S}$ is a sound rewriting of q with respect to $\mathcal{I}$ if for all source database $\mathcal{C}$ and for all global database $\mathcal{B} \in sem^\mathcal{C}(\mathcal{I})$, we have that $q_1^\mathcal{C} \subseteq q^\mathcal{B}$

From the above definition, it follows that a sound rewriting computes only certain answers: indeed, if q_1 is a sound rewriting, then for all source database $\mathcal{C}$,

$$q_1^\mathcal{C} \subseteq \bigcap \{q^\mathcal{B} \mid \forall\, \mathcal{B} \in sem^\mathcal{C}(\mathcal{I})\}$$

Outline

Perfect rewriting

What is the relationship between answering by rewriting and certain answers? [Calvanese & al. ICDT'05]:
Let us consider the "best possible" rewriting

Define $cert_{[q,\mathcal{I}]}(\cdot)$ to be the function that, with q and $\mathcal{I}$ fixed, given source database $\mathcal{C}$, computes the certain answers $cert(q, \mathcal{I}, \mathcal{C})$

- $cert_{[q,\mathcal{I}]}$ can be seen as a query on the alphabet $\mathcal{A}_\mathcal{S}$
- $cert_{[q,\mathcal{I}]}$ is a (sound) rewriting of q wrt $\mathcal{I}$, i.e., it computes only certain answers
- No sound rewriting exists that is better than $cert_{[q,\mathcal{I}]}$, i.e., if r is a sound rewriting of q wrt $\mathcal{I}$, then $r \subseteq cert_{[q,\mathcal{I}]}$

Hence, $cert_{[q,\mathcal{I}]}$ is called the perfect rewriting of q wrt $\mathcal{I}$

Perfect rewriting

What is the relationship between answering by rewriting and certain answers? [Calvanese & al. ICDT'05]:
Let us consider the "best possible" rewriting

Define $cert_{[q,\mathcal{I}]}(\cdot)$ to be the function that, with q and $\mathcal{I}$ fixed, given source database $\mathcal{C}$, computes the certain answers $cert(q, \mathcal{I}, \mathcal{C})$

- $cert_{[q,\mathcal{I}]}$ can be seen as a query on the alphabet $\mathcal{A}_\mathcal{S}$
- $cert_{[q,\mathcal{I}]}$ is a (sound) rewriting of q wrt $\mathcal{I}$, i.e., it computes only certain answers
- No sound rewriting exists that is better than $cert_{[q,\mathcal{I}]}$, i.e., if r is a sound rewriting of q wrt $\mathcal{I}$, then $r \subseteq cert_{[q,\mathcal{I}]}$

Hence, $cert_{[q,\mathcal{I}]}$ is called the perfect rewriting of q wrt $\mathcal{I}$

Perfect rewriting

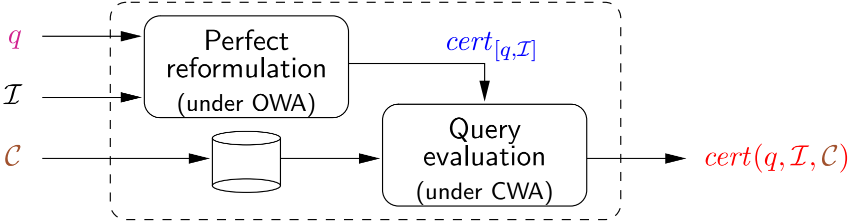
What is the relationship between answering by rewriting and certain answers? [Calvanese & al. ICDT'05]:
Let us consider the "best possible" rewriting

Define $cert_{[q,\mathcal{I}]}(\cdot)$ to be the function that, with q and $\mathcal{I}$ fixed, given source database $\mathcal{C}$, computes the certain answers $cert(q,\mathcal{I},\mathcal{C})$

- $cert_{[q,\mathcal{I}]}$ can be seen as a query on the alphabet $\mathcal{A}_{\mathcal{S}}$
- $cert_{[q,\mathcal{I}]}$ is a (sound) rewriting of q wrt $\mathcal{I}$, i.e., it computes only certain answers
- No sound rewriting exists that is better than $cert_{[q,\mathcal{I}]}$, i.e., if r is a sound rewriting of q wrt $\mathcal{I}$, then $r \subseteq cert_{[q,\mathcal{I}]}$

Hence, $cert_{[q,\mathcal{I}]}$ is called the perfect rewriting of q wrt $\mathcal{I}$

Query answering: reformulation + evaluation



In principle, we need an arbitrary query language to express $cert_{[q,\mathcal{I}]}$

More about rewriting

We are interested in rewritings r of q wrt $\mathcal{I}$ that are:

- sound, i.e., compute only tuples in $cert(q, \mathcal{I}, \mathcal{C})$ for every $\mathcal{C}$ (i.e., $r \subseteq cert_{[q,\mathcal{I}]}$)
- expressed in a given query language $\mathcal{L}$
- sound, and maximal for a class of queries $\mathcal{L}$
- perfect

A sound rewriting r of q wrt $\mathcal{I}$ is maximal for $\mathcal{L}$ if for all $r' \in \mathcal{L}$, $r' \subseteq cert_{[q,\mathcal{I}]}$ implies $r \not\subset r'$

More about rewriting

We are interested in rewritings r of q wrt $\mathcal{I}$ that are:

- sound, i.e., compute only tuples in $cert(q, \mathcal{I}, \mathcal{C})$ for every $\mathcal{C}$ (i.e., $r \subseteq cert_{[q,\mathcal{I}]}$)
- expressed in a given query language $\mathcal{L}$
- sound, and maximal for a class of queries $\mathcal{L}$
- perfect

A sound rewriting r of q wrt $\mathcal{I}$ is maximal for $\mathcal{L}$ if for all $r' \in \mathcal{L}$, $r' \subseteq cert_{[q,\mathcal{I}]}$ implies $r \not\subset r'$

More about rewriting

We are interested in rewritings r of q wrt $\mathcal{I}$ that are:

- sound, i.e., compute only tuples in $cert(q, \mathcal{I}, \mathcal{C})$ for every $\mathcal{C}$ (i.e., $r \subseteq cert_{[q,\mathcal{I}]}$)
- expressed in a given query language $\mathcal{L}$
- sound, and maximal for a class of queries $\mathcal{L}$
- perfect

A sound rewriting r of q wrt $\mathcal{I}$ is maximal for $\mathcal{L}$ if for all $r' \in \mathcal{L}$, $r' \subseteq cert_{[q,\mathcal{I}]}$ implies $r \not\subset r'$

More about rewriting

We are interested in rewritings r of q wrt $\mathcal{I}$ that are:

- sound, i.e., compute only tuples in $cert(q, \mathcal{I}, \mathcal{C})$ for every $\mathcal{C}$ (i.e., $r \subseteq cert_{[q,\mathcal{I}]}$)
- expressed in a given query language $\mathcal{L}$
- sound, and maximal for a class of queries $\mathcal{L}$
- perfect

A sound rewriting r of q wrt $\mathcal{I}$ is maximal for $\mathcal{L}$ if for all $r' \in \mathcal{L}$, $r' \subseteq cert_{[q,\mathcal{I}]}$ implies $r \not\subset r'$

More about rewriting

We are interested in rewritings r of q wrt $\mathcal{I}$ that are:

- sound, i.e., compute only tuples in $cert(q, \mathcal{I}, \mathcal{C})$ for every $\mathcal{C}$ (i.e., $r \subseteq cert_{[q,\mathcal{I}]}$)
- expressed in a given query language $\mathcal{L}$
- sound, and maximal for a class of queries $\mathcal{L}$
- perfect

A sound rewriting r of q wrt $\mathcal{I}$ is maximal for $\mathcal{L}$ if for all $r' \in \mathcal{L}$, $r' \subseteq cert_{[q,\mathcal{I}]}$ implies $r \not\subset r'$

Properties of the perfect rewriting

- Can the perfect rewriting be expressed in a certain query language?
- For a given class of queries, what is the relationship between a maximal rewriting and the perfect rewriting?
  - From a semantical point of view
  - From a computational point of view
- Which is the computational complexity of finding the perfect rewriting, and how big is it?
- Which is the computational complexity of evaluating the perfect rewriting?

Outline

LAV without constraints – Query answering via rewriting

Given a LAV data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a query q' over $\mathcal{S}$, $exp(q')$ is the query over $\mathcal{G}$ that is obtained by substituting every atom with the view that $\mathcal{M}$ associates to it.

Let q be a conjunctive query over $\mathcal{G}$, and q' a conjunctive query over $\mathcal{S}$. q' is a sound rewriting of q if and only if $exp(q') \subseteq q$.

We may be interested in exact rewritings, i.e., rewritings q' that are logically equivalent to the query, modulo $\mathcal{M}$ (i.e., $exp(q') \equiv q$). However, exact rewritings may not exist.

LAV without constraints – Query answering via rewriting

Given a LAV data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a query q' over $\mathcal{S}$, $exp(q')$ is the query over $\mathcal{G}$ that is obtained by substituting every atom with the view that $\mathcal{M}$ associates to it.

Let q be a conjunctive query over $\mathcal{G}$, and q' a conjunctive query over $\mathcal{S}$. q' is a sound rewriting of q if and only if $exp(q') \subseteq q$.

We may be interested in exact rewritings, i.e., rewritings q' that are logically equivalent to the query, modulo $\mathcal{M}$ (i.e., $exp(q') \equiv q$). However, exact rewritings may not exist.

Exercise 2

- Prove the following:
  Let $\mathcal{I}$ be a LAV data integration system without constraints in the global schema, let q be a conjunctive query over $\mathcal{G}$, and let q' be a conjunctive query over $\mathcal{S}$. q' is a sound rewriting of q if and only if $exp(q') \subseteq q$.
- Exhibit a LAV data integration system and a query q such that no exact rewriting of q exists with respect to $\mathcal{I}$.

LAV without constraints – Rewriting for conjunctive queries

Consider a LAV data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a query q over $\mathcal{G}$. Let q and the queries in $\mathcal{M}$ be conjunctive queries.

Theorem

If the body of q has n atoms, and q' is a maximal rewriting in the class of conjunctive queries, then q' has at most n atoms.

Sketch of the proof: Since q' is a rewriting of q, we have that $exp(q') \subseteq q$. Consider the homomorphism h from q to $exp(q')$. Each atom in q is mapped by h to at most one atom in $exp(q')$. If there are more than n atoms in q', then the expansion of some atom in q' is disjoint from the image of h, and then this atom can be removed from q' while preserving containment (i.e., q' is not maximal).

This provides us with an algorithm for computing the set of maximal conjunctive rewritings.

LAV without constraints – Rewriting for conjunctive queries

Let q' be the union of all maximal rewritings of q for the class of CQs

Theorem (Levy & al. PODS'95, Abiteboul & Duschka PODS'98)

- *q' is the maximal rewriting for the class of unions of conjunctive queries (UCQs)*
- *q' is the perfect rewriting of q wrt $\mathcal{I}$*
- *q' is a PTIME query (actually, LogSpace)*
- *q' is an exact rewriting (equivalent to q for each database $\mathcal{B}$ of $\mathcal{I}$), if an exact rewriting exists*

Does this "ideal situation" carry on to cases where q and $\mathcal{M}$ allow for union?

LAV without constraints – Rewriting for conjunctive queries

Let q' be the union of all maximal rewritings of q for the class of CQs

Theorem (Levy & al. PODS'95, Abiteboul & Duschka PODS'98)

- *q' is the maximal rewriting for the class of unions of conjunctive queries (UCQs)*
- *q' is the perfect rewriting of q wrt $\mathcal{I}$*
- *q' is a PTIME query (actually, LogSpace)*
- *q' is an exact rewriting (equivalent to q for each database $\mathcal{B}$ of $\mathcal{I}$), if an exact rewriting exists*

Does this "ideal situation" carry on to cases where q and $\mathcal{M}$ allow for union?

LAV without constraints – Rewriting for positive views

When queries over the global schema in the mapping contain union:

- Computing certain answering is coNP-complete in data complexity [van der Meyden TCS'93]
- Hence, the perfect rewriting $cert_{[q,\mathcal{I}]}$ is a coNP-complete query, and therefore cannot be expressed as a union of conjunctive query

We do not have the ideal situation we had for conjunctive queries

Exercise 3

Prove the following:

When queries over the global schema of a LAV data integration system without constraints contain union, computing certain answering is coNP-complete in data complexity

Exercise 4

Define an algorithm based on rewriting for computing the certain answers to conjunctive queries in GLAV data integration systems without constraints.

Outline

Inclusion dependencies (IDs)

An inclusion dependency (ID) states that the presence of a tuple $\vec{t_1}$ in a relation implies the presence of a tuple $\vec{t_2}$ in another relation, where $\vec{t_2}$ contains a projection of the values contained in $\vec{t_1}$

Syntax of inclusion dependencies

$$r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$$

with $i_1, \ldots, i_k$ components of r, and $j_1, \ldots, j_k$ components of s

Example

For r of arity 3 and s of arity 2, the ID $r[1] \subseteq s[2]$ corresponds to the FOL sentence

$$\forall x, y, w.\ r(x, y, w) \rightarrow \exists z.\ s(z, x)$$

Note: IDs are a special form of tuple-generating dependencies

Inclusion dependencies (IDs)

An inclusion dependency (ID) states that the presence of a tuple $\vec{t_1}$ in a relation implies the presence of a tuple $\vec{t_2}$ in another relation, where $\vec{t_2}$ contains a projection of the values contained in $\vec{t_1}$

Syntax of inclusion dependencies

$$r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$$

with $i_1, \ldots, i_k$ components of r, and $j_1, \ldots, j_k$ components of s

Example

For r of arity 3 and s of arity 2, the ID $r[1] \subseteq s[2]$ corresponds to the FOL sentence

$$\forall x, y, w.\ r(x, y, w) \rightarrow \exists z.\ s(z, x)$$

Note: IDs are a special form of tuple-generating dependencies

Inclusion dependencies (IDs)

An inclusion dependency (ID) states that the presence of a tuple $\vec{t}_1$ in a relation implies the presence of a tuple $\vec{t}_2$ in another relation, where $\vec{t}_2$ contains a projection of the values contained in $\vec{t}_1$

Syntax of inclusion dependencies

$$r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$$

with $i_1, \ldots, i_k$ components of r, and $j_1, \ldots, j_k$ components of s

Example

For r of arity 3 and s of arity 2, the ID $r[1] \subseteq s[2]$ corresponds to the FOL sentence

$$\forall x, y, w.\ r(x, y, w) \rightarrow \exists z.\ s(z, x)$$

Note: IDs are a special form of tuple-generating dependencies

Inclusion dependencies – Example

Global schema $\mathcal{G}$: player$(Pname, YOB, Pteam)$
team$(Tname, Tcity, Tleader)$

Constraints: team$[Tleader, Tname]$ $\subseteq$ player$[Pname, Pteam]$

Sources $\mathcal{S}$: s_1 and s_3 store players
s_2 stores teams

Mapping $\mathcal{M}$: $\forall x, y, z \; s_1(x, y, z) \vee s_3(x, y, z) \;\rightarrow\;$ player(x, y, z)
$\forall x, y, z \; s_2(x, y, z) \;\rightarrow\;$ team(x, y, z)

Inclusion dependencies – Example retrieved global db

Source database $\mathcal{C}$:

s_1:

Totti	1971	Roma

s_2:

Juve	Torino	Del Piero

s_3:

Buffon	1978	Juve

Retrieved global database $\mathcal{M}(\mathcal{C})$:

player:

Totti	1971	Roma
Buffon	1978	Juve

team:

Juve	Torino	Del Piero

Inclusion dependencies – Example retrieved global db

Source database $\mathcal{C}$:

s_1:

Totti	1971	Roma

s_2:

Juve	Torino	Del Piero

s_3:

Buffon	1978	Juve

Retrieved global database $\mathcal{M}(\mathcal{C})$:

player:

Totti	1971	Roma
Buffon	1978	Juve

team:

Juve	Torino	Del Piero

Inclusion dependencies – Example retrieved global db

player:

Totti	1971	Roma
Buffon	1978	Juve
Del Piero	α	Juve

team:

Juve	Torino	Del Piero

The ID on the global schema tells us that Del Piero is a player of Juve

All global databases satisfying $\mathcal{I}$ have at least the tuples shown above, where α is some value of the domain Δ

Warnings

1. There may be an infinite number of databases satisfying $\mathcal{I}$
2. In case of cyclic IDs, databases satisfying $\mathcal{I}$ may be of infinite size

Inclusion dependencies – Example retrieved global db

player:

Totti	1971	Roma
Buffon	1978	Juve
Del Piero	α	Juve

team:

Juve	Torino	Del Piero

The ID on the global schema tells us that Del Piero is a player of Juve

All global databases satisfying $\mathcal{I}$ have at least the tuples shown above, where α is some value of the domain Δ

Warnings

1. There may be an infinite number of databases satisfying $\mathcal{I}$
2. In case of cyclic IDs, databases satisfying $\mathcal{I}$ may be of infinite size

Inclusion dependencies – Example retrieved global db

player:

Totti	1971	Roma
Buffon	1978	Juve
Del Piero	α	Juve

team:

Juve	Torino	Del Piero

The ID on the global schema tells us that Del Piero is a player of Juve

All global databases satisfying $\mathcal{I}$ have at least the tuples shown above, where α is some value of the domain Δ

Warnings

1. There may be an infinite number of databases satisfying $\mathcal{I}$
2. In case of cyclic IDs, databases satisfying $\mathcal{I}$ may be of infinite size

Chasing inclusion dependencies – Infinite construction

Intuitive strategy: Add new facts until IDs are satisfied

Problem: Infinite construction in the presence of cyclic IDs

Example

Let r be binary with
$r[2] \subseteq r[1]$

Suppose $\mathcal{M}(\mathcal{C}) = \{\ r(a,b)\ \}$

1. add $r(b, c_1)$
2. add $r(c_1, c_2)$
3. add $r(c_2, c_3)$
4. ... (ad infinitum)

Example

Let r, s be binary with
$r[1] \subseteq s[1], \quad s[2] \subseteq r[1]$

Suppose $\mathcal{M}(\mathcal{C}) = \{\ r(a,b)\ \}$

1. add $s(a, c_1)$
2. add $r(c_1, c_2)$
3. add $s(c_1, c_3)$
4. add $r(c_3, c_4)$
5. ... (ad infinitum)

Chasing inclusion dependencies – Infinite construction

Intuitive strategy: Add new facts until IDs are satisfied

Problem: Infinite construction in the presence of cyclic IDs

Example

Let r be binary with

$r[2] \subseteq r[1]$

Suppose $\mathcal{M}(\mathcal{C}) = \{\ r(a,b)\ \}$

1. add $r(b,c_1)$
2. add $r(c_1,c_2)$
3. add $r(c_2,c_3)$
4. ... (ad infinitum)

Example

Let r, s be binary with

$r[1] \subseteq s[1], \quad s[2] \subseteq r[1]$

Suppose $\mathcal{M}(\mathcal{C}) = \{\ r(a,b)\ \}$

1. add $s(a,c_1)$
2. add $r(c_1,c_2)$
3. add $s(c_1,c_3)$
4. add $r(c_3,c_4)$
5. ... (ad infinitum)

Chasing inclusion dependencies – Infinite construction

Intuitive strategy: Add new facts until IDs are satisfied

Problem: Infinite construction in the presence of cyclic IDs

Example

Let r be binary with
$r[2] \subseteq r[1]$

Suppose $\mathcal{M}(\mathcal{C}) = \{ r(a,b) \}$

1. add $r(b, c_1)$
2. add $r(c_1, c_2)$
3. add $r(c_2, c_3)$
4. ... (ad infinitum)

Example

Let r, s be binary with
$r[1] \subseteq s[1], \quad s[2] \subseteq r[1]$

Suppose $\mathcal{M}(\mathcal{C}) = \{ r(a,b) \}$

1. add $s(a, c_1)$
2. add $r(c_1, c_2)$
3. add $s(c_1, c_3)$
4. add $r(c_3, c_4)$
5. ... (ad infinitum)

The ID-chase rule

The chase for IDs has only one rule, the ID-chase rule

Let $\mathcal{D}$ be a database:

if the schema contains the ID $r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$
and there is a fact in $\mathcal{D}$ of the form $r(a_1, \ldots, a_n)$
and there are no facts in $\mathcal{D}$ of the form $s(b_1, \ldots, b_m)$
such that $a_{i_\ell} = b_{j_\ell}$ for each $\ell \in \{1, \ldots, k\}$,
then add to $\mathcal{D}$ the fact $s(c_1, \ldots, c_m)$,
where for each $h \in \{1, \ldots, m\}$,
if $h = j_\ell$ for some ℓ then $c_h = a_{i_\ell}$
otherwise c_h is a new constant symbol (not in $\mathcal{D}$ yet)

Notice: New existential symbols are introduced (skolem terms)

Properties of the chase

- Bad news: the chase is in general infinite
- Good news: the chase identifies a canonical database (with variables)
- We can use the chase to prove soundness and completeness of a query processing method
- . . . but only for positive queries!

Limiting the chase

Why don't we use a finite number of existential constants in the chase?

Example

Consider $r[1] \subseteq s[1]$, and $s[2] \subseteq r[1]$ and suppose $\mathcal{M}(\mathcal{C}) = \{\ r(a,b)\ \}$

Compute chase($\mathcal{M}(\mathcal{C})$) with only one new constant c_1:

0) $r(a,b)$; 1) add $s(a,c_1)$ 2) add $r(c_1,c_1)$ 3) add $s(c_1,c_1)$

This database is not a canonical database for $\mathcal{I}$ wrt $\mathcal{C}$
E.g., for query $q = \{\ (x) \mid r(x,y), s(y,y)\ \}$, we have $a \in q^{\text{chase}(\mathcal{M}(\mathcal{C}))}$ while $a \notin cert(q,\mathcal{I},\mathcal{C})$

Arbitrarily limiting the chase is unsound, for any finite number of new constants

Rewriting: Chasing the query

- Instead of chasing the data, we chase the query
- Is the dual notion of the database chase
- IDs are applied from right to left to the query atoms
- Advantage: much easier termination conditions, which imply:
  - decidability properties
  - efficiency

> This technique provides an algorithm for rewriting UCQs under IDs

Rewriting rule for inclusion dependencies

Intuition: Use the IDs as basic rewriting rules

Example

Consider a query $q = \{ (x,z) \mid \text{player}(x,y,z) \}$

and the constraint $\text{team}[Tleader, Tname] \subseteq \text{player}[Pname, Pteam]$

as a logic rule: $\text{player}(w_3, w_4, w_1) \leftarrow \text{team}(w_1, w_2, w_3)$

We add to the rewriting the query $q' = \{ (x,z) \mid \text{team}(x,y,z) \}$

Definition

Basic rewriting step:

when an atom unifies with the head of the rule

substitute the atom with the body of the rule

Rewriting rule for inclusion dependencies

Intuition: Use the IDs as basic rewriting rules

Example

Consider a query $q = \{ (x, z) \mid \text{player}(x, y, z) \}$

and the constraint $\text{team}[Tleader, Tname] \subseteq \text{player}[Pname, Pteam]$

as a logic rule: $\text{player}(w_3, w_4, w_1) \leftarrow \text{team}(w_1, w_2, w_3)$

We add to the rewriting the query $q' = \{ (x, z) \mid \text{team}(x, y, z) \}$

Definition

Basic rewriting step:

when an atom unifies with the head of the rule

substitute the atom with the body of the rule

Rewriting rule for inclusion dependencies

Intuition: Use the IDs as basic rewriting rules

Example

Consider a query $q = \{ (x, z) \mid \text{player}(x, y, z) \}$

and the constraint $\text{team}[Tleader, Tname] \subseteq \text{player}[Pname, Pteam]$

as a logic rule: $\text{player}(w_3, w_4, w_1) \leftarrow \text{team}(w_1, w_2, w_3)$

We add to the rewriting the query $q' = \{ (x, z) \mid \text{team}(x, y, z) \}$

Definition

Basic rewriting step:

when an atom unifies with the head of the rule

substitute the atom with the body of the rule

Query Rewriting for IDs – Algorithm *ID-rewrite*

Iterative execution of:

1. Reduction:
   - Atoms that unify with other atoms are eliminated and the unification is applied
   - Variables that appear only once are marked
2. Basic rewriting step
   - A rewriting step is applicable to an atom if it does not eliminate variables that appear somewhere else
   - May introduce fresh variables

Note: The algorithm works directly for unions of conjunctive queries (UCQs), and produces an UCQ as result

Query Rewriting for IDs – Algorithm *ID-rewrite*

Iterative execution of:

1. Reduction:
   - Atoms that unify with other atoms are eliminated and the unification is applied
   - Variables that appear only once are marked
2. Basic rewriting step
   - A rewriting step is applicable to an atom if it does not eliminate variables that appear somewhere else
   - May introduce fresh variables

Note: The algorithm works directly for unions of conjunctive queries (UCQs), and produces an UCQ as result

The algorithm *ID-rewrite*

Input: relational schema $\mathcal{G}$, set Ψ_{ID} of IDs, UCQ Q
Output: perfect rewriting of Q
$Q' := Q$;
repeat
  $Q_{aux} := Q'$;
  for each $q \in Q_{aux}$ **do**
  (a) **for each** $g_1, g_2 \in \mathit{body}(q)$ **do**
    if g_1 and g_2 unify **then** $Q' := Q' \cup \{\tau(\mathit{reduce}(q, g_1, g_2))\}$;
  (b) **for each** $g \in \mathit{body}(q)$ **do**
    for each $ID \in \Psi_{ID}$ **do**
      if ID is applicable to g
        then $Q' := Q' \cup \{\, q[g/\mathit{rewrite}(g, ID)]\,\}$
until $Q_{aux} = Q'$;
return Q'

Query answering in GAV under IDs

Properties of *ID-rewrite*

- *ID-rewrite* terminates
- *ID-rewrite* produces a perfect rewriting of the input query

More precisely, let $\mathit{unf}_{\mathcal{M}}(q)$ be the unfolding of the query q wrt the GAV mapping $\mathcal{M}$

Theorem

$\mathit{unf}_{\mathcal{M}}(\text{ID-rewrite}(q))$ is a perfect rewriting of the query q

Theorem

Query answering in GAV systems under IDs is in PTIME in data complexity (actually in LOGSPACE)

Query answering in GAV under IDs

Properties of *ID-rewrite*

- *ID-rewrite* terminates
- *ID-rewrite* produces a perfect rewriting of the input query

More precisely, let $\mathit{unf}_{\mathcal{M}}(q)$ be the unfolding of the query q wrt the GAV mapping $\mathcal{M}$

Theorem

$\mathit{unf}_{\mathcal{M}}(\mathit{ID\text{-}rewrite}(q))$ is a perfect rewriting of the query q

Theorem

Query answering in GAV systems under IDs is in PTIME *in data complexity (actually in* LOGSPACE*)*

Exercise 5

An exclusion dependency (ED) states that the presence of a tuple $\vec{t_1}$ in a relation implies the absence of a tuple $\vec{t_2}$ in another relation, where $\vec{t_2}$ contains a projection of the values contained in $\vec{t_1}$

Syntax of exclusion dependencies

$$r[i_1, \ldots, i_k] \cap s[j_1, \ldots, j_k] = \emptyset$$

with $i_1, \ldots, i_k$ components of r, and $j_1, \ldots, j_k$ components of s

Find an algorithm for computing certain answers to conjunctive queries in GAV with inclusion and exclusion dependencies.

Outline

Query answering based on counterexample

Given $\mathcal{I}$, $\mathcal{C}$, q, and $\vec{t}$, a counterexample to $\vec{t} \in cert(q, \mathcal{I}, \mathcal{C})$ is a database $\mathcal{B} \in sem^{\mathcal{C}}(\mathcal{I})$ such that $\vec{t} \notin q^{\mathcal{B}}$

Thus, query answering based on counterexample can be described as follows:

Given $\mathcal{I}$, $\mathcal{C}$, q, and $\vec{t}$, check whether there exists a counterexample to $\vec{t} \in cert(q, \mathcal{I}, \mathcal{C})$

Exercise 6

Consider the case of LAV with positive views.

- $\vec{t} \notin cert(q, \mathcal{I}, \mathcal{C})$ iff there is a database $\mathcal{B}_1 \in sem^{\mathcal{C}}(\mathcal{I})$ such that $\vec{t} \notin q^{\mathcal{B}_1}$
- In LAV with positive views, the mapping $\mathcal{M}$ has the form:

$$\forall \vec{x}.\ \phi_{\mathcal{S}}(\vec{x}) \ \rightarrow \ \exists \vec{y}_1.\ \alpha_1(\vec{x}, \vec{y}_1) \ \vee \ \cdots \ \vee \ \exists \vec{y}_h\ \alpha_h(\vec{x}, \vec{y}_h))$$

Find an algorithm for computing certain answers to conjuntive queries in LAV with positive views

Outline

Query containment under constraints

Definition

Query containment (under constraints) is the problem of checking whether $q_1^{\mathcal{D}}$ is contained in $q_2^{\mathcal{D}}$ for every database $\mathcal{D}$ (satisfying the constraints), where q_1, q_2 are queries of the same arity

Exercise 7

How can we solve the problem of computing the certain answers in terms of containment?

Part III

Beyond relational data

Outline

9 Semi-structured data integration
- Semi-structured data and queries
- Graph databases

10 Ontology-based data integration

Outline

Outline

Introduction to semi-structured data integration

The global schema (and possibly the sources) is expressed in a formalism aimed at modeling data with more flexibility wrt the relational model

There are at least two types of semi-structured data models

- Graph databases
  Talk 18 – Paolo Guagliardo "View-based query processing"
- XML data
  Talk 14 – Lucja Kot, "XML data integration"

Outline

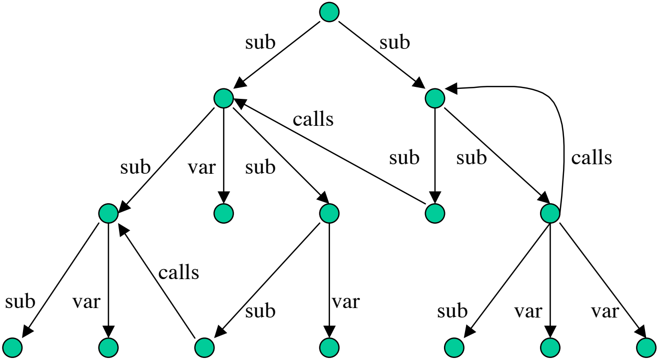
Graph databases

A graph database is a finite directed graph whose edges are labeled with a given finite alphabet Σ.

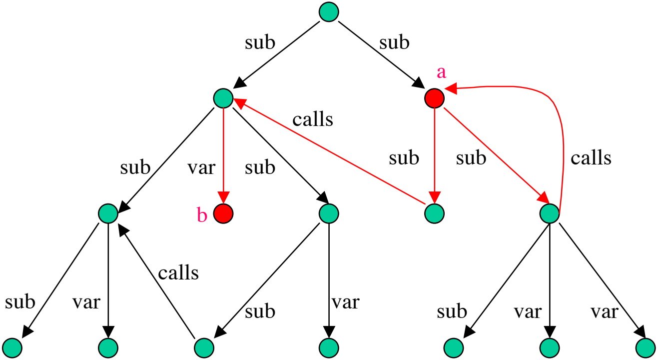
Each node represents an object, and an edge from x to y labeled r represents the fact that the relation r holds between x and y.

The basic query language for graph databases is the language of regular path queries. A regular path query (RPQ) over Σ is defined in terms of a regular language over Σ. The answer $Q(D)$ to an RPQ Q over a grapg database D is the set of pairs of objects connected in D by a path traversing a sequence of edges forming a word in the regular language $L(Q)$ defined by Q.

Global semi-structured database

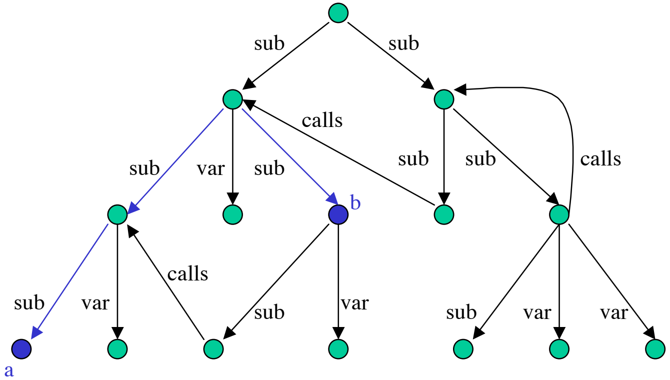


Global semi-structured databases and queries



Regular Path Query (RPQ): $(sub)^* \cdot (sub \cdot (calls \cup sub))^* \cdot var$

Global semi-structured databases and queries



2RPQ: $(sub^{-})^{*} \cdot (var \cup sub)$

The case of RPQ with LAV mappings

Given

- $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where
  - $\mathcal{G}$ simply fixes the labels (alphabet Σ) of a semi-structured database
  - the sources in $\mathcal{S}$ are binary relations
  - the mapping $\mathcal{M}$ is of type LAV, and associates to each source s a 2RPQ w over Σ

$$\forall x, y \quad s(x, y) \subseteq x \xrightarrow{w} y$$

- a source database $\mathcal{C}$
- a 2RPQ Q over Σ
- a pair of objects $\vec{t}$

we want to determine whether $\vec{t} \in cert(Q, \mathcal{I}, \mathcal{C})$.

Query answering: Technique

- We search for a counterexample to $\vec{t} \in cert(Q, \mathcal{I}, \mathcal{C})$, i.e., a database $\mathcal{B} \in sem^{\mathcal{C}}(\mathcal{I})$ such that $\vec{t} \notin cert(Q, \mathcal{I}, \mathcal{C})$
- Crucial point: it is sufficient to restrict our attention to canonical databases, i.e., databases $\mathcal{B}$ that can be represented by a word $w_{\mathcal{B}}$

$$\$\, \mathbf{d_1\, w_1\, d_2}\, \$\, \mathbf{d_3\, w_2\, d_4}\, \$ \cdots \$\, \mathbf{d_{2m-1}\, w_m\, d_{2m}}\, \$$$

where $d_1, \ldots, d_{2m}$ are constants in $\mathcal{C}$, $w_i \in \Sigma^+$, and \$ acts as a separator

⇒ **Use word-automata theoretic techniques!** [Calvanese & al. PODS 2000]

Query answering: Technique

To check whether $(c, d) \notin cert(Q, \mathcal{I}, \mathcal{C})$, we check for nonemptiness of A, that is the intersection of

- the one-way automaton A_0 that accepts words that represent databases, i.e., words of the form $(\$ \cdot \mathcal{C} \cdot \Sigma^+ \cdot \mathcal{C})^* \cdot \$$
- the one-way automata corresponding to the various $A_{(S_i,a,b)}$ (for each source S_i and for each pair $(a, b) \in S_i^{\mathcal{C}}$)
- the one-way automaton corresponding to the complement of $A_{(Q,c,d)}$

Indeed, any word accepted by such intersection automaton represents a counterexample to $(c, d) \in cert(Q, \mathcal{I}, \mathcal{C})$.

Query answering: Complexity

- All two-way automata constructed above are of linear size in the size of Q, the queries associated to $S_1, \ldots, S_k$, and $S_1^{\mathcal{C}}, \ldots, S_k^{\mathcal{C}}$. Hence, the corresponding one-way automata would be exponential.
- However, we do not need to construct A explicitly. Instead, we can construct it **on the fly** while checking for nonemptiness.

Query answering for 2RPQs is PSPACE-complete in combined complexity, and coNP-complete in data complexity.

Outline

The use of ontologies in data integration

The global schema is expressed as an ontology, aimed at modeling the domain of discourse from a conceptual point of view, in turn expressed in termis of logic.

Description Logics (DLs) [Baader & al. 2003] are logics specifically designed to represent and reason on structured knowledge. The domain of interest is composed of objects and is structured into:

- concepts, which correspond to classes, and denote sets of objects
- roles, which correspond to (binary) relationships, and denote binary relations on objects

The knowledge is asserted through so-called assertions, i.e., logical axioms.

Brief history of Description Logics

1977 *KL-ONE* Workshop: from Semantic Networks and Frames to Description Logics

1984 Trade-off expressiveness – complexity of inference [Brachman & al. 1984]

1986 Description logics for conceptual modeling

1989 *Classic* system – polynomial inference, but no assertions

1990 Expressive DLs – tableaux correspondence with modal logic and PDLs automata

1995 Conceptual models fully captured in DLs

1998 Optimized tableaux make expressive DLs practical Query answering in DLs

2000 Standardization efforts – OIL, DAML+OIL, OWL, OWL2

2005 Polynomial DLs with assertions – $\mathcal{EL}$, *DL-Lite*

Ingredients of a Description Logic

A DL is characterized by:

1. A description language: how to form concepts and roles
   $\mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists\mathsf{hasChild} \sqcap \forall\mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
2. A mechanism to specify knowledge about concepts and roles (i.e., a TBox)
   $$\mathcal{T} = \{ \begin{array}{lcl} \mathsf{Father} & \equiv & \mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists\mathsf{hasChild}, \\ \mathsf{HappyFather} & \sqsubseteq & \mathsf{Father} \sqcap \forall\mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer}) \end{array} \}$$
3. A mechanism to specify properties of objects (i.e., an ABox)
   $\mathcal{A} = \{ \mathsf{HappyFather}(\mathtt{john}), \quad \mathsf{hasChild}(\mathtt{john}, \mathtt{mary}) \}$
4. A set of inference services: how to reason on a given KB
   $\mathcal{T} \models \mathsf{HappyFather} \sqsubseteq \exists\mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
   $\mathcal{T} \cup \mathcal{A} \models (\mathsf{Doctor} \sqcup \mathsf{Lawyer})(\mathtt{mary})$

Description language

A description language provides the means for defining:

- concepts, corresponding to classes: interpreted as sets of objects;
- roles, corresponding to relationships: interpreted as binary relations on objects.

To define concepts and roles:

- We start from a (finite) alphabet of atomic concepts and atomic roles, i.e., simply names for concept and roles.
- Then, by applying specific constructors, we can build complex concepts and roles, starting from the atomic ones.

A description language is characterized by the set of constructs that are available for that.

Semantics of a description language

The formal semantics of DLs is given in terms of interpretations.

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of:

- a nonempty set $\Delta^{\mathcal{I}}$, the domain of $\mathcal{I}$
- an interpretation function $\cdot^{\mathcal{I}}$, which maps
  - each individual a to an element $a^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$
  - each atomic concept A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$
  - each atomic role P to a subset $P^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

The interpretation function is extended to complex concepts and roles according to their syntactic structure.

Concept constructors

Construct	Syntax	Example	Semantics
atomic concept	A	Doctor	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic role	P	hasChild	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
atomic negation	$\neg A$	¬Doctor	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
conjunction	$C \sqcap D$	Hum ⊓ Male	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
(unqual.) exist. res.	$\exists R$	∃hasChild	$\{ a \mid \exists b. (a,b) \in R^{\mathcal{I}} \}$
value restriction	$\forall R.C$	∀hasChild.Male	$\{a \mid \forall b. (a,b) \in R^{\mathcal{I}} \to b \in C^{\mathcal{I}}\}$
bottom	$\bot$		$\emptyset$

(C, D denote arbitrary concepts and R an arbitrary role)

The above constructs form the basic language $\mathcal{AL}$ of the family of $\mathcal{AL}$ languages.

Concept constructors

Construct	Syntax	Example	Semantics
atomic concept	A	Doctor	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic role	P	hasChild	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
atomic negation	$\neg A$	¬Doctor	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
conjunction	$C \sqcap D$	Hum ⊓ Male	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
(unqual.) exist. res.	$\exists R$	∃hasChild	$\{ a \mid \exists b. (a, b) \in R^{\mathcal{I}} \}$
value restriction	$\forall R.C$	∀hasChild.Male	$\{a \mid \forall b. (a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
bottom	$\bot$		$\emptyset$

(C, D denote arbitrary concepts and R an arbitrary role)

The above constructs form the basic language $\mathcal{AL}$ of the family of $\mathcal{AL}$ languages.

Further examples of DL constructs

- Disjunction $\mathcal{U}$: $\mathsf{Doctor} \sqcup \mathsf{Lawyer}$
- Qualified existential restriction $\mathcal{E}$: $\exists\mathsf{hasChild}.\mathsf{Doctor}$
- Full negation $\mathcal{C}$: $\neg(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
- Number restrictions $\mathcal{N}$: $(\geq 2\,\mathsf{hasChild})$ $(\leq 1\,\mathsf{sibling})$
- Qualified number restrictions $\mathcal{Q}$: $(\geq 2\,\mathsf{hasChild}.\,\mathsf{Doctor})$
- Inverse role $\mathcal{I}$: $\exists\mathsf{hasChild}^{-}.\mathsf{Doctor}$
- Reflexive-transitive role closure $_{reg}$: $\exists\mathsf{hasChild}^{*}.\mathsf{Doctor}$

Structural properties vs. asserted properties

We have seen how to build complex concept and roles expressions, which allow one to denote classes with a complex structure.

However, in order to represent real world domains, one needs the ability to assert properties of classes and relationships between them (e.g., as done in UML class diagrams).

The assertion of properties is done in DLs by means of an ontology.

Description Logics ontology

Is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox:

The TBox consists of a set of assertions on concepts and roles:

- Inclusion assertions on concepts: $C_1 \sqsubseteq C_2$
- Inclusion assertions on roles: $R_1 \sqsubseteq R_2$
- Property assertions on (atomic) roles:

  (**transitive** P) (**symmetric** P) (**domain** P C)
  (**functional** P) (**reflexive** P) (**range** P C) $\cdots$

The ABox consists of a set of membership assertions on individuals:

- for concepts: $A(c)$
- for roles: $P(c_1, c_2)$ (we use c_i to denote individuals)

Description Logics ontology

Is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox:

The TBox consists of a set of assertions on concepts and roles:

- Inclusion assertions on concepts: $C_1 \sqsubseteq C_2$
- Inclusion assertions on roles: $R_1 \sqsubseteq R_2$
- Property assertions on (atomic) roles:
  (**transitive** P) (**symmetric** P) (**domain** P C)
  (**functional** P) (**reflexive** P) (**range** P C) $\cdots$

The ABox consists of a set of membership assertions on individuals:

- for concepts: $A(c)$
- for roles: $P(c_1, c_2)$ (we use c_i to denote individuals)

Description Logics ontology – Example

Note: We use $C_1 \equiv C_2$ as an abbreviation for $C_1 \sqsubseteq C_2,\ C_2 \sqsubseteq C_1$.

TBox assertions:

- Inclusion assertions on concepts:

$$\begin{array}{rcl} \text{Father} & \equiv & \text{Human} \sqcap \text{Male} \sqcap \exists\text{hasChild} \\ \text{HappyFather} & \sqsubseteq & \text{Father} \sqcap \forall\text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{Happy}) \\ \text{HappyAnc} & \sqsubseteq & \forall\text{descendant}.\text{HappyFather} \\ \text{Teacher} & \sqsubseteq & \neg\text{Doctor} \sqcap \neg\text{Lawyer} \end{array}$$

- Inclusion assertions on roles:

$$\text{hasChild} \sqsubseteq \text{descendant} \qquad \text{hasFather} \sqsubseteq \text{hasChild}^-$$

- Property assertions on roles:

(**transitive** descendant), (**reflexive** descendant), (**functional** hasFather)

ABox membership assertions:

- Teacher(mary), hasFather(mary, john), HappyAnc(john)

Description Logics ontology – Example

Note: We use $C_1 \equiv C_2$ as an abbreviation for $C_1 \sqsubseteq C_2,\ C_2 \sqsubseteq C_1$.

TBox assertions:

- Inclusion assertions on concepts:

$$\begin{array}{rcl} \text{Father} & \equiv & \text{Human} \sqcap \text{Male} \sqcap \exists\text{hasChild} \\ \text{HappyFather} & \sqsubseteq & \text{Father} \sqcap \forall\text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{Happy}) \\ \text{HappyAnc} & \sqsubseteq & \forall\text{descendant}.\text{HappyFather} \\ \text{Teacher} & \sqsubseteq & \neg\text{Doctor} \sqcap \neg\text{Lawyer} \end{array}$$

- Inclusion assertions on roles:

$$\text{hasChild} \sqsubseteq \text{descendant} \qquad \text{hasFather} \sqsubseteq \text{hasChild}^-$$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant),
  (**functional** hasFather)

ABox membership assertions:

- Teacher(mary), hasFather(mary, john), HappyAnc(john)

Semantics of a Description Logics ontology

The semantics is given by specifying when an interpretation $\mathcal{I}$ satisfies an assertion:

- $C_1 \sqsubseteq C_2$ is satisfied by $\mathcal{I}$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- $R_1 \sqsubseteq R_2$ is satisfied by $\mathcal{I}$ if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$.
- A property assertion (**prop** P) is satisfied by $\mathcal{I}$ if $P^{\mathcal{I}}$ is a relation that has the property **prop**.

- $A(c)$ is satisfied by $\mathcal{I}$ if $c^{\mathcal{I}} \in A^{\mathcal{I}}$.
- $P(c_1, c_2)$ is satisfied by $\mathcal{I}$ if $(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.

This leads to the notion of model of a DL ontology. An interpretation $\mathcal{I}$ is a model of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ if it satisfies all assertions in $\mathcal{T}$ and all assertions in $\mathcal{A}$.

Semantics of a Description Logics ontology

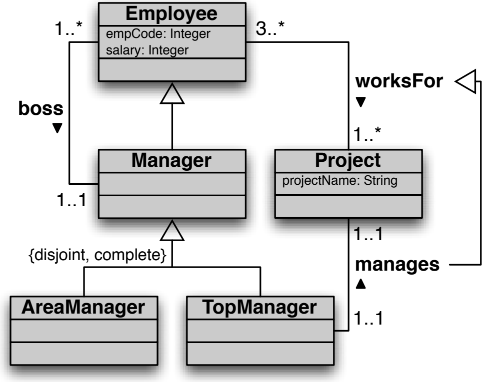
The semantics is given by specifying when an interpretation $\mathcal{I}$ satisfies an assertion:

- $C_1 \sqsubseteq C_2$ is satisfied by $\mathcal{I}$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- $R_1 \sqsubseteq R_2$ is satisfied by $\mathcal{I}$ if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$.
- A property assertion (**prop** P) is satisfied by $\mathcal{I}$ if $P^{\mathcal{I}}$ is a relation that has the property **prop**.

- $A(c)$ is satisfied by $\mathcal{I}$ if $c^{\mathcal{I}} \in A^{\mathcal{I}}$.
- $P(c_1, c_2)$ is satisfied by $\mathcal{I}$ if $(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.

This leads to the notion of model of a DL ontology. An interpretation $\mathcal{I}$ is a model of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ if it satisfies all assertions in $\mathcal{T}$ and all assertions in $\mathcal{A}$.

Example



$$
\begin{array}{rcl}
\mathsf{Manager} & \sqsubseteq & \mathsf{Employee} \\
\mathsf{AreaManager} & \sqsubseteq & \mathsf{Manager} \\
\mathsf{TopManager} & \sqsubseteq & \mathsf{Manager} \\
\mathsf{Manager} & \sqsubseteq & \mathsf{AreaManager} \sqcup \\
 & & \mathsf{TopManager} \\
\mathsf{AreaManager} & \sqsubseteq & \neg\mathsf{TopManager} \\
\mathsf{Employee} & \sqsubseteq & \exists \mathsf{salary} \\
\exists \mathsf{salary}^- & \sqsubseteq & \mathsf{Integer} \\
\exists \mathsf{worksFor} & \sqsubseteq & \mathsf{Employee} \\
\exists \mathsf{worksFor}^- & \sqsubseteq & \mathsf{Project} \\
\mathsf{Employee} & \sqsubseteq & \exists \mathsf{worksFor} \\
\mathsf{Project} & \sqsubseteq & (\geq 3\, \mathsf{worksFor}^-) \\
\end{array}
$$

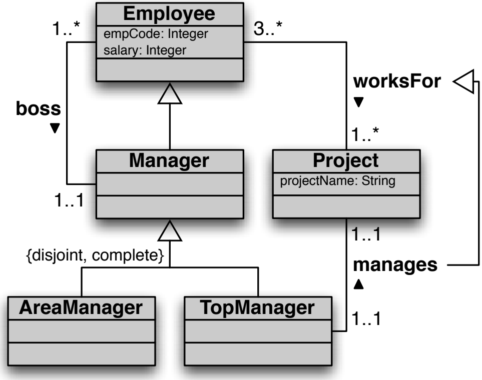
$$(\mathbf{funct}\ \mathsf{manages})$$
$$(\mathbf{funct}\ \mathsf{manages}^-)$$

$$\mathsf{manages} \sqsubseteq \mathsf{worksFor}$$

$$\ldots$$

Note: Domain and range of associations are expressed by means of concept inclusions.

Example



$$
\begin{array}{rcl}
\mathsf{Manager} & \sqsubseteq & \mathsf{Employee} \\
\mathsf{AreaManager} & \sqsubseteq & \mathsf{Manager} \\
\mathsf{TopManager} & \sqsubseteq & \mathsf{Manager} \\
\mathsf{Manager} & \sqsubseteq & \mathsf{AreaManager} \sqcup \\
 & & \mathsf{TopManager} \\
\mathsf{AreaManager} & \sqsubseteq & \neg\mathsf{TopManager} \\
\mathsf{Employee} & \sqsubseteq & \exists\mathsf{salary} \\
\exists\mathsf{salary}^- & \sqsubseteq & \mathsf{Integer} \\
\exists\mathsf{worksFor} & \sqsubseteq & \mathsf{Employee} \\
\exists\mathsf{worksFor}^- & \sqsubseteq & \mathsf{Project} \\
\mathsf{Employee} & \sqsubseteq & \exists\mathsf{worksFor} \\
\mathsf{Project} & \sqsubseteq & (\geq 3\, \mathsf{worksFor}^-)
\end{array}
$$

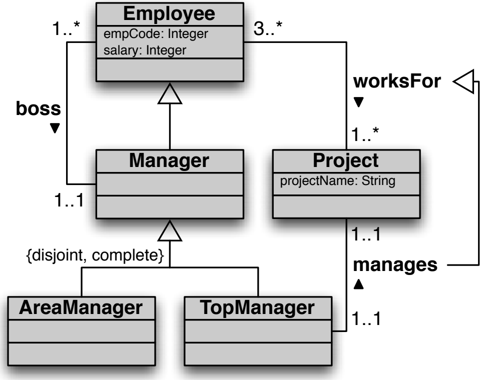
$$(\textbf{funct}\ \mathsf{manages})$$
$$(\textbf{funct}\ \mathsf{manages}^-)$$

$$\mathsf{manages} \sqsubseteq \mathsf{worksFor}$$

...

Note: Domain and range of associations are expressed by means of concept inclusions.

Example



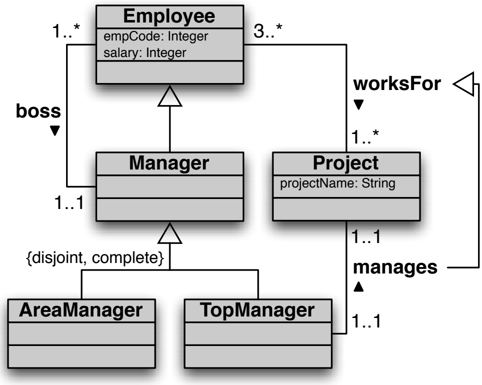
$$\begin{array}{rcl}
\text{Manager} & \sqsubseteq & \text{Employee} \\
\text{AreaManager} & \sqsubseteq & \text{Manager} \\
\text{TopManager} & \sqsubseteq & \text{Manager} \\
\text{Manager} & \sqsubseteq & \text{AreaManager} \sqcup \\
 & & \text{TopManager} \\
\text{AreaManager} & \sqsubseteq & \neg\text{TopManager} \\
\text{Employee} & \sqsubseteq & \exists\text{salary} \\
\exists\text{salary}^{-} & \sqsubseteq & \text{Integer} \\
\exists\text{worksFor} & \sqsubseteq & \text{Employee} \\
\exists\text{worksFor}^{-} & \sqsubseteq & \text{Project} \\
\text{Employee} & \sqsubseteq & \exists\text{worksFor} \\
\text{Project} & \sqsubseteq & (\geq 3\,\text{worksFor}^{-})
\end{array}$$

$$(\textbf{funct}\ \text{manages})$$
$$(\textbf{funct}\ \text{manages}^{-})$$

$$\begin{array}{rcl}
\text{manages} & \sqsubseteq & \text{worksFor} \\
 & \cdots &
\end{array}$$

Note: Domain and range of associations are expressed by means of concept inclusions.

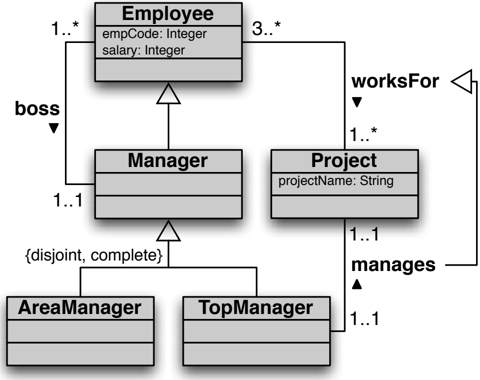
Example



$$\begin{array}{rcl}
\text{Manager} & \sqsubseteq & \text{Employee} \\
\text{AreaManager} & \sqsubseteq & \text{Manager} \\
\text{TopManager} & \sqsubseteq & \text{Manager} \\
\text{Manager} & \sqsubseteq & \text{AreaManager} \sqcup \\
 & & \text{TopManager} \\
\text{AreaManager} & \sqsubseteq & \neg\text{TopManager} \\
\text{Employee} & \sqsubseteq & \exists\text{salary} \\
\exists\text{salary}^- & \sqsubseteq & \text{Integer} \\
\exists\text{worksFor} & \sqsubseteq & \text{Employee} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{Project} \\
\text{Employee} & \sqsubseteq & \exists\text{worksFor} \\
\text{Project} & \sqsubseteq & (\geq 3\ \text{worksFor}^-) \\
 & (\textbf{funct}\ \text{manages}) & \\
 & (\textbf{funct}\ \text{manages}^-) & \\
\text{manages} & \sqsubseteq & \text{worksFor} \\
 & \ldots &
\end{array}$$

Note: Domain and range of associations are expressed by means of concept inclusions.

Example



$$
\begin{array}{rcl}
\text{Manager} & \sqsubseteq & \text{Employee} \\
\text{AreaManager} & \sqsubseteq & \text{Manager} \\
\text{TopManager} & \sqsubseteq & \text{Manager} \\
\text{Manager} & \sqsubseteq & \text{AreaManager} \sqcup \\
 & & \text{TopManager} \\
\text{AreaManager} & \sqsubseteq & \neg\text{TopManager} \\
\text{Employee} & \sqsubseteq & \exists\text{salary} \\
\exists\text{salary}^- & \sqsubseteq & \text{Integer} \\
\exists\text{worksFor} & \sqsubseteq & \text{Employee} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{Project} \\
\text{Employee} & \sqsubseteq & \exists\text{worksFor} \\
\text{Project} & \sqsubseteq & (\geq 3\ \text{worksFor}^-)
\end{array}
$$

$$(\textbf{funct}\ \text{manages})$$

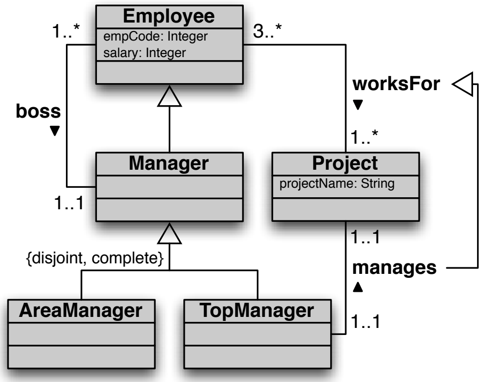
$$(\textbf{funct}\ \text{manages}^-)$$

$$\text{manages} \sqsubseteq \text{worksFor}$$

$$\ldots$$

Note: Domain and range of associations are expressed by means of concept inclusions.

Example



$$\begin{array}{rcl}
\text{Manager} & \sqsubseteq & \text{Employee} \\
\text{AreaManager} & \sqsubseteq & \text{Manager} \\
\text{TopManager} & \sqsubseteq & \text{Manager} \\
\text{Manager} & \sqsubseteq & \text{AreaManager} \sqcup \text{TopManager} \\
\text{AreaManager} & \sqsubseteq & \neg \text{TopManager} \\
\text{Employee} & \sqsubseteq & \exists \text{salary} \\
\exists \text{salary}^- & \sqsubseteq & \text{Integer} \\
\exists \text{worksFor} & \sqsubseteq & \text{Employee} \\
\exists \text{worksFor}^- & \sqsubseteq & \text{Project} \\
\text{Employee} & \sqsubseteq & \exists \text{worksFor} \\
\text{Project} & \sqsubseteq & (\geq 3\, \text{worksFor}^-) \\
\end{array}$$

$$(\textbf{funct}\ \text{manages})$$

$$(\textbf{funct}\ \text{manages}^-)$$

$$\text{manages} \sqsubseteq \text{worksFor}$$

$$\cdots$$

Note: Domain and range of associations are expressed by means of concept inclusions.

TBox reasoning

- Concept Satisfiability:
  C is satisfiable wrt $\mathcal{T}$, if $C^{\mathcal{I}}$ is not empty for some model $\mathcal{I}$ of $\mathcal{T}$.

- Subsumption:
  C_1 is subsumed by C_2 wrt $\mathcal{T}$, if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.

- Equivalence:
  C_1 and C_2 are equivalent wrt $\mathcal{T}$, if $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.

- Disjointness:
  C_1 and C_2 are disjoint wrt $\mathcal{T}$, if $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$.

Analogous definitions hold for role satisfiability, subsumption, equivalence, and disjointness.

Reasoning over an ontology

- Ontology Satisfiability: Verify whether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.
- Concept Instance Checking: Verify whether an individual c is an instance of a concept C in every model of $\mathcal{O}$.
- Role Instance Checking: Verify whether a pair (c_1, c_2) of individuals is an instance of a role R in every model of $\mathcal{O}$.
- Query Answering: see later ...

Reasoning in Description Logics – Example

TBox:

- Inclusion assertions on concepts:

$$\begin{array}{rcl} \mathsf{Father} & \equiv & \mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild} \\ \mathsf{HappyFather} & \sqsubseteq & \mathsf{Father} \sqcap \forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer} \sqcup \mathsf{Happy}) \\ \mathsf{HappyAnc} & \sqsubseteq & \forall \mathsf{descendant}.\mathsf{HappyFather} \\ \mathsf{Teacher} & \sqsubseteq & \neg \mathsf{Doctor} \sqcap \neg \mathsf{Lawyer} \end{array}$$

- Inclusion assertions on roles:

$$\mathsf{hasChild} \sqsubseteq \mathsf{descendant} \qquad \mathsf{hasFather} \sqsubseteq \mathsf{hasChild}^-$$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant),
  (**functional** hasFather)

The above TBox logically implies: $\mathsf{HappyAncestor} \sqsubseteq \mathsf{Father}$.

ABox:

- $\mathsf{Teacher}(\mathtt{mary})$, $\mathsf{hasFather}(\mathtt{mary}, \mathtt{john})$, $\mathsf{HappyAnc}(\mathtt{john})$

The above TBox and ABox logically imply: $\mathsf{Happy}(\mathtt{mary})$

Reasoning in Description Logics – Example

TBox:

- Inclusion assertions on concepts:

$$\begin{array}{rcl} \text{Father} & \equiv & \text{Human} \sqcap \text{Male} \sqcap \exists\text{hasChild} \\ \text{HappyFather} & \sqsubseteq & \text{Father} \sqcap \forall\text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{Happy}) \\ \text{HappyAnc} & \sqsubseteq & \forall\text{descendant}.\text{HappyFather} \\ \text{Teacher} & \sqsubseteq & \neg\text{Doctor} \sqcap \neg\text{Lawyer} \end{array}$$

- Inclusion assertions on roles:

$\text{hasChild} \sqsubseteq \text{descendant}$ $\qquad$ $\text{hasFather} \sqsubseteq \text{hasChild}^-$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant),
  (**functional** hasFather)

The above TBox logically implies: $\text{HappyAncestor} \sqsubseteq \text{Father}$.

ABox:

- $\text{Teacher}(\texttt{mary})$, $\text{hasFather}(\texttt{mary}, \texttt{john})$, $\text{HappyAnc}(\texttt{john})$

The above TBox and ABox logically imply: $\text{Happy}(\texttt{mary})$

Reasoning in Description Logics – Example

TBox:

- Inclusion assertions on concepts:

$$\begin{array}{rcl} \text{Father} & \equiv & \text{Human} \sqcap \text{Male} \sqcap \exists \text{hasChild} \\ \text{HappyFather} & \sqsubseteq & \text{Father} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{Happy}) \\ \text{HappyAnc} & \sqsubseteq & \forall \text{descendant}.\text{HappyFather} \\ \text{Teacher} & \sqsubseteq & \neg \text{Doctor} \sqcap \neg \text{Lawyer} \end{array}$$

- Inclusion assertions on roles:

$\text{hasChild} \sqsubseteq \text{descendant}$ $\quad\quad$ $\text{hasFather} \sqsubseteq \text{hasChild}^-$

- Property assertions on roles:

(**transitive** descendant), (**reflexive** descendant),
(**functional** hasFather)

The above TBox logically implies: $\text{HappyAncestor} \sqsubseteq \text{Father}$.

ABox:

- $\text{Teacher}(\texttt{mary})$, $\text{hasFather}(\texttt{mary}, \texttt{john})$, $\text{HappyAnc}(\texttt{john})$

The above TBox and ABox logically imply: $\text{Happy}(\texttt{mary})$

Reasoning in Description Logics – Example

TBox:

- Inclusion assertions on concepts:

$$\begin{aligned} \text{Father} &\equiv \text{Human} \sqcap \text{Male} \sqcap \exists\text{hasChild} \\ \text{HappyFather} &\sqsubseteq \text{Father} \sqcap \forall\text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{Happy}) \\ \text{HappyAnc} &\sqsubseteq \forall\text{descendant}.\text{HappyFather} \\ \text{Teacher} &\sqsubseteq \neg\text{Doctor} \sqcap \neg\text{Lawyer} \end{aligned}$$

- Inclusion assertions on roles:

$$\text{hasChild} \sqsubseteq \text{descendant} \qquad \text{hasFather} \sqsubseteq \text{hasChild}^-$$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant),
  (**functional** hasFather)

The above TBox logically implies: $\text{HappyAncestor} \sqsubseteq \text{Father}$.

ABox:

- $\text{Teacher}(\texttt{mary})$, $\text{hasFather}(\texttt{mary}, \texttt{john})$, $\text{HappyAnc}(\texttt{john})$

The above TBox and ABox logically imply: $\text{Happy}(\texttt{mary})$

Complexity of reasoning over DL ontologies

TBox reasoning over DL ontologies is in general complex:

- TBox reasoning over ontologies in virtually all traditional DLs is EXPTIME-hard
- Stays in EXPTIME even in the most expressive DLs (except when using nominals, i.e., `ObjectOneOf`).
- There are TBox reasoners that perform reasonably well in practice for such DLs (e.g, Racer, Pellet, Fact++, ...)

Queries over Description Logics ontologies

If we want to use ontologies as global schemas in data integration, we have to allow for queries expressed over a DL ontology

A conjunctive query $q(\vec{x})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ has the form $q(\vec{x}) \leftarrow \exists \vec{y}.\, conj(\vec{x}, \vec{y})$ where $conj(\vec{x}, \vec{y})$ is a conjunction of atoms which

- has as predicate symbol an atomic concept or role of $\mathcal{T}$, and
- may use variables and constants that are individuals in $\mathcal{A}$

The certain answers to $q(\vec{x})$ over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, denoted $cert(q, \mathcal{O})$ are the tuples $\vec{c}$ of constants such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{O}$.

DLs must be restricted considerably if we want tractable conjunctive query answering (even when the complexity is measured wrt the size of the ABox only)

Related talks at DEIS'10

- Talk 2 – Piotr Wieczorek, "Query answering in data integration"
- Talk 7 – Slawomir Staworko, "Consistent query answering"
- Talk 8 – Yazmin A. Ibanez, "Description logics for data integration"
- Talk 9 – Ekaterini Ioannou, "Data cleaning for data integration"
- Talk 10 – Armin Roth, "Peer data management systems"
- Talk 11 – Sebastian Skritek, "Theory of Peer Data Management"
- Talk 14 – Lucja Kot, "XML data integration"
- Talk 18 – Paolo Guagliardo "View-based query processing"
- Talk 22 – Marie Jacob, "Learning and discovering queries and mappings"