Query answering over UML class diagrams

Giuseppe De Giacomo

Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma

Outline

Outline

Incomplete information and query answering

- Incomplete information in data: missing / unknown / partially specified data
- Query answering
  - Over usual databases (complete information):
    QA by evaluation (or "model checking")
    $$D \models Q$$
    i.e., D is seen as an interpretation (for simplicity we assume the query to be boolean, no free variables)
  - Over incomplete databases (incomplete information):
    QA by logical implication (or "entailment")
    $$\forall \mathcal{I}. \mathcal{I} \models D \text{ implies } \mathcal{I} \models Q$$

Incomplete databases

A common form of incomplete databases are the so-called "naive tables", which include values and "labelled nulls" (standing for unknown values) [IL84].

Example

Employee

name
Smith
$null_1$
Brown

Manager

mgr	*mgd*
Smith	$null_1$
$null_1$	Brown
Brown	$null_2$

- *Const*: we have infinite constants, corresponding to domain objects as usual;
- *Nulls*: we have a countably infinite set of nulls, corresponding to variables ranging over *Cons*;
- Tables are incomplete, i.e., more tuples may belong to them, corresponding to the so called "open-world-assumption" or OWA. (For example $null_2$ belongs to *Employee* though not reported in the table.)

Incomplete databases: semantics

Semantics of incomplete databases:

- A valuation function for nulls is a assignment function $\sigma : Nulls \rightarrow Const$ (essentially nulls are considered as individual variables in logic).
- We denote by $\mathcal{I}, \sigma \models D$ the fact that for every tuple $(t_1, \ldots, t_n) \in P$ for each table P we have $\mathcal{I}, \sigma \models P(t_1, \ldots, t_n)$.
- We define in logic the set of databases completing D as

$$Models(D) \;=\; \{\mathcal{I} | \text{ there exists a } \sigma \text{ such that } \mathcal{I}, \sigma \models D\}$$

Example

Employee

name
Smith
$null_1$
Brown

Manager

mgr	*mgd*
Smith	$null_1$
$null_1$	Brown
Brown	$null_2$

Employee

name
Smith
White
Brown
Black

Manager

mgr	*mgd*
Smith	White
White	Brown
Brown	Black

Employee

name
Smith
Brown
Black
. . .

Manager

mgr	*mgd*
Smith	Brown
Brown	Brown
Brown	Brown

. . .

Certain answers to a query

An incomplete database acts like a logical theory: it selects models.

Query answering in complete databases

The answer to a query $q(\vec{x})$ over a complete database D, denoted q^D, is the set of tuples $\vec{c}$ of constants of $Const$ such that the $\vec{c} \in q^D$ is to true in D.

Query answering in incomplete databases

The certain answer to a query $q(\vec{x})$ over an incomplete database D, denoted $cert(q, D)$, is the set of tuples $\vec{c}$ of constants of $Const$ such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of D.

Note:

- It q is boolean, and D is incomplete: we write $D \models q$ iff q evaluates to true in every model $\mathcal{I}$ of D, (otherwise we write $D \not\models q$.
- We use the same notation as for query answering based on evaluation: the difference is in the incompleteness of the database.

Query languages for incomplete databases

Which query language to use?

1. Full SQL (or equivalently, first-order logic)
   - NO: in the presence of incomplete information, query answering becomes undecidable (FOL validity).
     (Notice this holds already for an empty incomplete database!)
2. Conjunctive queries (or better union of conjunctive queries)
   - Conjunctive queries are well behaved wrt containment. Can they be used for query answering in presence of incomplete information.
     YES! See what follows.

Outline

Conjunctive queries and incomplete databases

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

$$q(\vec{x}) \leftarrow \exists \vec{y}. R_1(\vec{x}, \vec{y}) \wedge \cdots \wedge R_k(\vec{x}, \vec{y})$$

where each $R_i(\vec{x}, \vec{y})$ is an atom using (some of) the free variables $\vec{x}$, the existentially quantified variables $\vec{y}$, and possibly constants.

We will also use the simpler Datalog notation:

$$q(\vec{x}) \leftarrow R_1(\vec{x}, \vec{y}), \ldots, R_k(\vec{x}, \vec{y})$$

Note:

- CQs contain no disjunction, no negation, no universal quantification.
- Correspond to SQL/relational algebra select-project-join (SPJ) queries – the most frequently asked queries.
- A Boolean CQ is a CQ without free variables $\Rightarrow q() \leftarrow \exists \vec{y}. R_1(\vec{y}) \wedge \cdots \wedge R_k(\vec{y})$.

Conjunctive queries and incomplete databases

Containment of conjunctive queries $q_1 \subseteq q_2$ is decidable: and LOGSPACE in q_1 and NP-complete in q_2 [ChandraMerlin77].

Given an incomplete database D as above we can construct in linear time a (boolean) conjunctive query q_D that fully captures it.

- For each tuple in a table of D becomes an atom in the conjunctive query q_D.
- For each labelled nulls occurring in D becomes an existentially quantified variable in q_D.

Example

$E(mployee)$

name
Smith
$null_1$
Brown

$M(anager)$

mgr	*mgd*
Smith	$null_1$
$null_1$	Brown
Brown	$null_2$

$$\exists x_1, x_2.E(Smith) \wedge E(x_1) \wedge E(Brown) \wedge M(Smith, x_1) \wedge M(x_1, Brown) \wedge M(Brown, x_2)$$

Conjunctive queries and incomplete databases

Theorem ([IL84])

Let D be a database with incomplete information as above (naive tables), q_D the corresponding conjunctive query constructed as above, and q a boolean (union) of conjunctive query. Then:

$$D \models q \text{ iff } q_D \subseteq q$$

Proof.

For the first "iff":

1. Observe that the models of D by construction coincide with that of the formula q_D: that is $\forall \mathcal{I}.\mathcal{I} \models D$ iff $\mathcal{I} \models q_D$.
2. Moreover, $q_D \subseteq q$ in the case of boolean queries stands for $\forall \mathcal{I}.\mathcal{I} \models q_D$ implies $\mathcal{I} \models q$, or simply $q_D \models q$.
3. Hence, by (1) $D \models q$ iff $q_D \models q$. □

Conjunctive queries and incomplete databases

Also by [ChandraMerlin77] we get:

Theorem ([IL84])

Let D be a database with incomplete information as above (naive tables), q_D the corresponding conjunctive query constructed as above, $\mathcal{I}_{q_D}$ its canonical database, and q a boolean (union) of conjunctive query. Then:

$$D \models q \text{ iff } \mathcal{I}_{q_D} \models q$$

Note: $\mathcal{I}_{q_D}$ is exactly D with nulls interpreted as additional constants!

Hence:

Compute certain answers of non boolean CQs over incomplete databases

Given a non boolean (U)CQ q and an incomplete database D:

1. Evaluate q over D as it was a complete database
2. filter out all answers where null appears (certain answers are constituted by tuples of constants in *Const*)

Conjunctive queries and incomplete databases

As a consequence of the above theorem we have:

Computing certain answers for (union) of conjunctive queries over databases with incomplete information (naive tables) is:

- LOGSPACE in data complexity
- NP-complete in query complexity and combined complexity

Note1: Exactly as for the case of complete information!

Note2: Use of CQs is crucial, since for full FOL we get undecidability!

Examples of CQs over an incomplete database

Example

$E(mployee)$

name
Smith
$null_1$
Brown

$M(anager)$

mgr	*mgd*
Smith	$null_1$
$null_1$	Brown
Brown	$null_2$

- Queries:
$$q_1(x,y) \leftarrow M(x,y)$$
$$q_2(x) \leftarrow \exists y.M(x,y)$$
$$q_3(x) \leftarrow \exists y_1,y_2,y_3.M(x,y_1) \land M(y_1,y_2) \land M(y_2,y_3)$$
$$q_4(x,y_3) \leftarrow \exists y_1,y_2.M(x,y_1) \land M(y_1,y_2) \land M(y_2,y_3)$$
- Answers:
q_1: { }
q_2: { Smith, Brown }
q_3: { Smith }
q_4: { }

Outline

UML Class Diagram

An UML class diagram

- Captured by a finite set of logical axioms that describe universal properties (i.e., properties of all objects belonging to classes/associations).
- Represents intensional knowledge
- Corresponds to schema level information in database terms
- Corresponds to a set of constraints on class and association memberships
- Describes the semantics of the objects
- Corresponds to "TBox" (or the so-called proper "ontology") in ontological languages which are often used instead of FOL (e.g., Descripton Logics, see later)

(Possibly partial or incomplete) instantiation

A (possibly partial or incomplete) instantiation aka object diagram (i.e., properties of single objects or relationships between them)

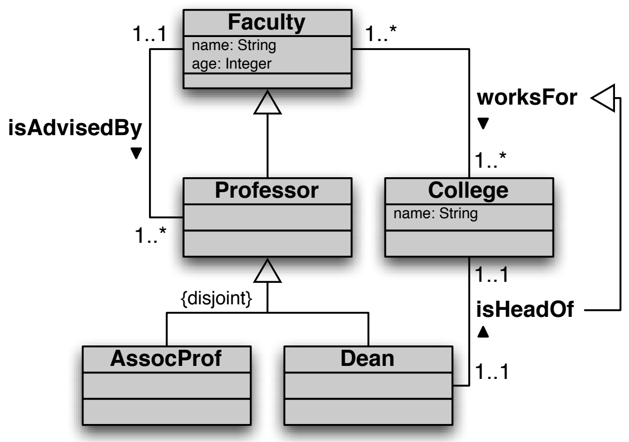
- Captured by a finite set of atomic facts in logic
- Represents extensional knowledge
- Corresponds to instance level information in database terms
- Corresponds to (incomplete) database in databases (though under constraints!)
- Describes actual data
- Correspond to "ABox" in ontological languages.

Knowledge Bases

We call knowledge base (KB) or sometime ontology the logical theory obtained by the union of the set of FOL formulas $\mathcal{T}$ and $\mathcal{A}$ where:

- $\mathcal{T}$ is the "TBox" and is formed by the formulas capturing the UML class diagram
- $\mathcal{A}$ is the "ABox" and is formed by the facts capturing the (possibly partial or incomplete) instantiation

Example of a query over a KB



TBox

$$
\begin{array}{rcl}
\forall x.\mathrm{Professor}(x) & \rightarrow & \mathrm{Faculty}(x) \\
\forall x.\mathrm{AssocProf}(x) & \rightarrow & \mathrm{Professor}(x) \\
\forall x.\mathrm{Dean}(x) & \rightarrow & \mathrm{Professor}(x) \\
\forall x.\mathrm{AssocProf}(x) & \rightarrow & \neg\mathrm{Dean}(x) \\
\forall x.\mathrm{Faculty}(x) & \rightarrow & \exists y.\mathrm{age}(x, y) \\
\forall x.\exists y.\mathrm{age}(y, x) & \rightarrow & \mathrm{Integer}(x) \\
\forall x.\exists y.\mathrm{worksFor}(x, y) & \rightarrow & \mathrm{Faculty}(x) \\
\forall x.\exists y.\mathrm{worksFor}(y, x) & \rightarrow & \mathrm{College}(x) \\
\forall x.\mathrm{Faculty}(x) & \rightarrow & \exists y.\mathrm{worksFor}(x, y) \\
\forall x.\mathrm{College}(x) & \rightarrow & \exists y.\mathrm{worksFor}(y, x) \\
& \cdots &
\end{array}
$$

ABox

$\cdots$

Query: *(note: in the case of incomplete information, we need to focus on (U)CQs because full FOL is undecidable even without intensional knowledge)*

$$
q(nf, af, nd) \leftarrow \exists f, c, d, ad.\ \mathrm{worksFor}(f, c) \wedge \mathrm{isHeadOf}(d, c) \wedge \mathrm{name}(f, nf) \wedge \mathrm{name}(d, nd) \wedge \mathrm{age}(f, af) \wedge \mathrm{age}(d, ad) \wedge af = ad
$$

Query answering under different assumptions

There are fundamentally different assumptions when addressing query answering in presence of a KB:

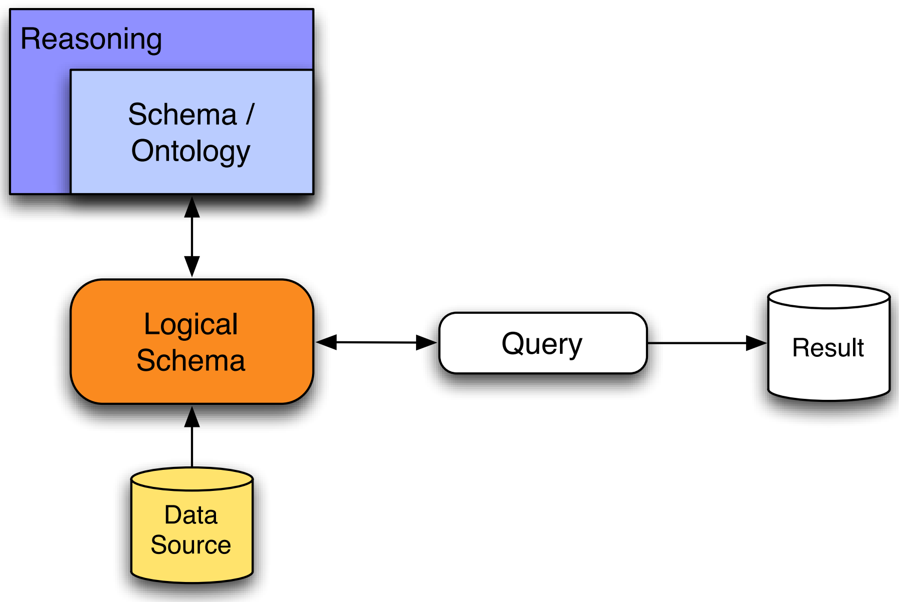
- Traditional database (DB) assumption:
  - Studied in mainly in Databases.
  - Data are complete (CWA).
  - Intensional knowledge/schema not used in query answering.
  - Query answering based on evaluation.
- Knowledge representation (KR) assumption:
  - Studied in mainly in Artificial Intelligence.
  - Assumes incompleteness in the data (incomplete databases) (OWA).
  - Intensional knowledge/schema must be used in query answering.
  - Query answering based on logical implication.

Query answering under the DB assumption

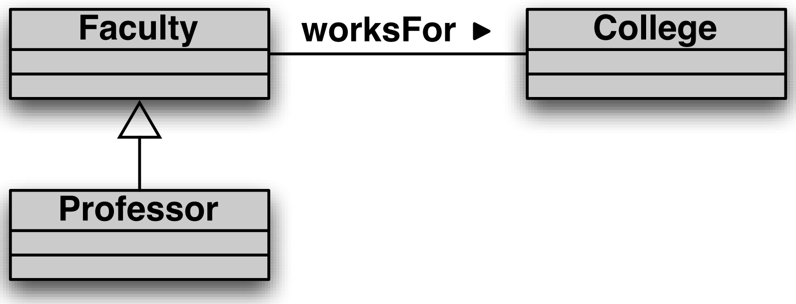
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- During query answering the data is assumed to satisfy the schema, and therefore the schema is not used.

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

Query answering under the DB assumption



Query answering under the DB assumption: example



For each class/property we have a (complete) table in the database.

DB: Faculty = { john, mary, paul }
Professor = { john, paul }
College = { collA, collB }
worksFor = { (john,collA), (mary,collB) }

Query: $q(x) \leftarrow \exists c.\, \text{Professor}(x), \text{College}(c), \text{worksFor}(x, c)$

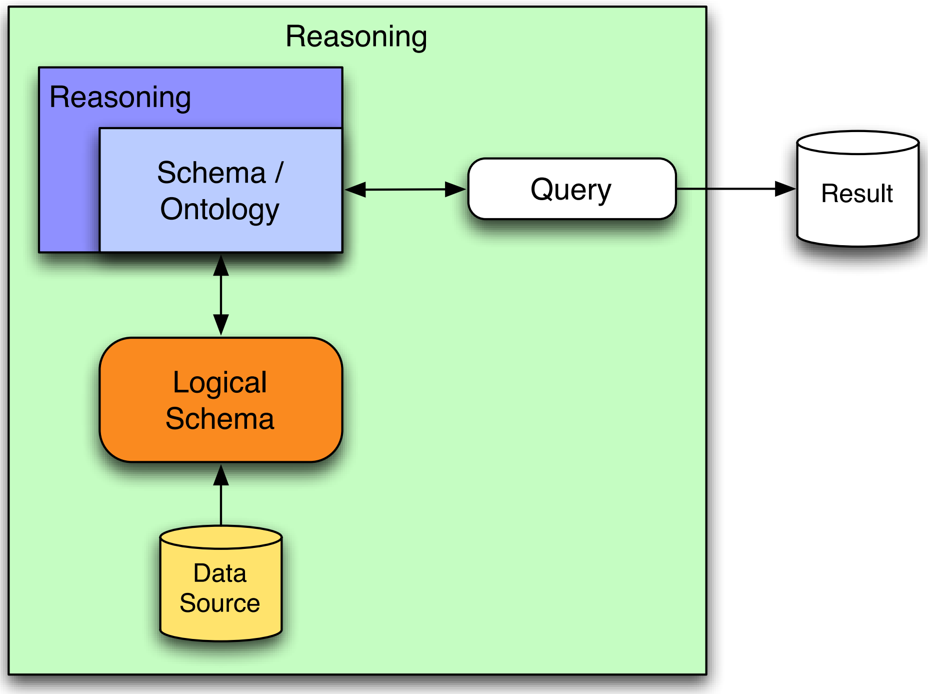
Answer: { john }

Query answering under the KR assumption

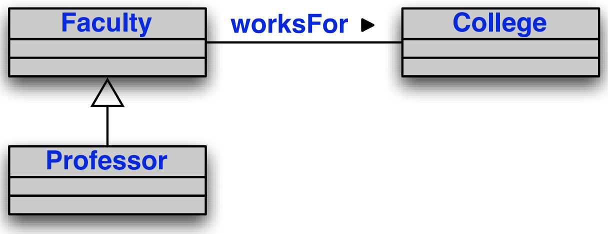
- The TBox imposes constraints on the data.
- Actual data (ABox) may be incomplete w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness.

$\leadsto$ Query answering amounts to logical inference, which is computationally much more costly in general.

Query answering under the KR assumption



Query answering under the KR assumption: example



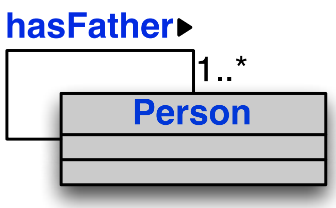
The tables in the database may be incompletely specified, or even missing for some classes/properties.

DB: Professor ⊇ { john, paul }
College ⊇ { collA, collB }
worksFor ⊇ { (john,collA), (mary,collB) }

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

Answer: { john, paul, mary }

Query answering under the KR assumption: another example



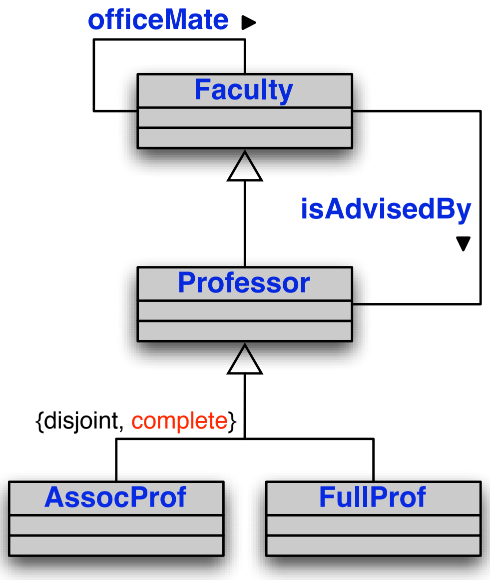
Each person has a father, who is a person.

DB: Person ⊇ { john, paul, toni }
hasFather ⊇ { (john,paul), (paul,toni) }

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

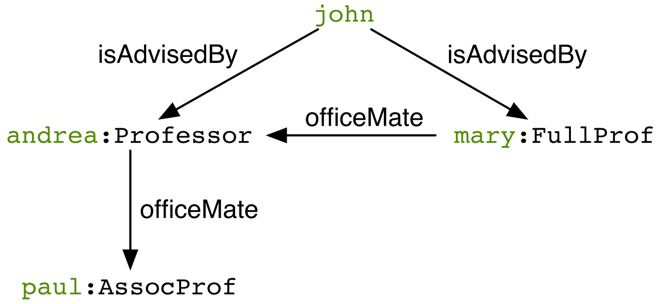
Answers: to q_1: { (john,paul), (paul,toni) }
to q_2: { john, paul, toni }
to q_3: { john, paul, toni }
to q_4: { }

QA under the KR assumption: Andrea's example

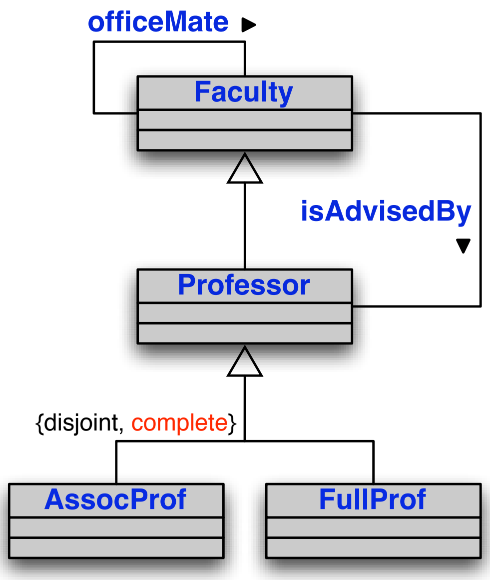


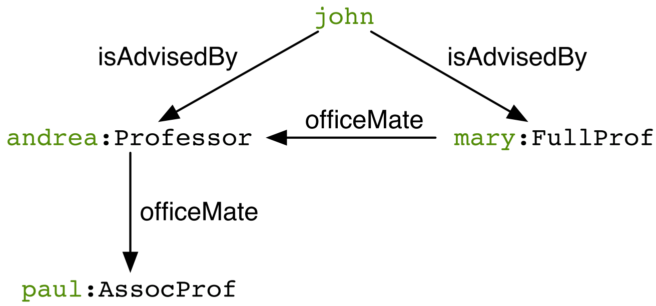
$\text{Professor} \equiv \text{AssocProf} \sqcup \text{FullProf}$

$\text{Faculty} \supseteq \{\ \texttt{andrea},\ \texttt{paul},\ \texttt{mary},\ \texttt{john}\ \}$
$\text{Professor} \supseteq \{\ \texttt{andrea},\ \texttt{paul},\ \texttt{mary}\ \}$
$\text{AssocProf} \supseteq \{\ \texttt{paul}\ \}$
$\text{FullProf} \supseteq \{\ \texttt{mary}\ \}$
$\text{isAdvisedBy} \supseteq \{\ (\texttt{john},\texttt{andrea}),\ (\texttt{john},\texttt{mary})\ \}$
$\text{officeMate} \supseteq \{\ (\texttt{mary},\texttt{andrea}),\ (\texttt{andrea},\texttt{paul})\ \}$



QA under the KR assumption – Andrea's example





$$q() \leftarrow \exists y, z.\ \text{isAdvisedBy}(\texttt{john}, y),\ \text{FullProf}(y),\ \text{officeMate}(y, z),\ \text{AssocProf}(z)$$

Answer: yes!

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

Query answering when accessing data through KBs

We have to face the difficulties of both DB and KB assumptions:

- The actual data is stored in external information sources (i.e., databases), and thus its size is typically very large.
- The KB 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 KB.
- 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.

Certain answers to a query

Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an KB (aka an ontology), $\mathcal{I}$ an interpretation for $\mathcal{K}$, and $q(\vec{x})$ a query.

Def.: The answer to $q(\vec{x})$ over $\mathcal{I}$ (model of $\mathcal{K}$), denoted $q^{\mathcal{I}}$

... is the set of tuples $\vec{c}$ of constants such that the formula $q(\vec{x})$ evaluates to true in $\mathcal{I}$.

We are interested in finding those answers that hold in all models of an KB.

Def.: The certain answers to $q(\vec{x})$ over $\mathcal{K}$, denoted $cert(q, \mathcal{K})$

... are the tuples $\vec{c}$ of constants such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{K}$.

Note: when q is boolean, we write $\mathcal{K} \models q$ iff q evaluates to true in every model $\mathcal{I}$ of $\mathcal{K}$, $\mathcal{K} \not\models q$ otherwise.

Data complexity

Various parameters affect the complexity of query answering over a KB.

Depending on which parameters we consider, we get different complexity measures:

- Data complexity: only the size of the ABox (i.e., the data) matters. TBox and query are considered fixed.
- Query complexity: only the size of the query matters. TBox and ABox are considered fixed.
- Schema complexity: only the size of the TBox (i.e., the schema) matters. ABox and query are considered fixed.
- Combined complexity: no parameter is considered fixed.

Typically the size of the data largely dominates the size of the conceptual layer (and of the query).
$\rightsquigarrow$ Data complexity is the relevant complexity measure.

Complexity of query answering in KBs

QA has been studied extensively for (unions of) CQs in the context of Description Logic-based ontology languages, which can be though of as specific FOL formalisms for class-based representation (cf. UML class diagrams or ER):

CQ query answering	Combined complexity	Data complexity
Complete databases	NP-complete	in LogSpace (1)
Incomplete databases (naive tables, OWA) no TBox	NP-complete	in LogSpace (1)
UML Class Diagrams or OWL2* TBoxes	2ExpTime-hard	coNP-hard (2)

** OWL 2 is a W3C standard based on Description Logics (DLs).*

(1) This is what we need to scale with the data.
(2) Already for a TBox with a single disjunction (see Andrea's example).

Questions

- Can we find interesting logics for the TBox for which the query answering problem can be solved efficiently (i.e., in LogSpace)?
- If yes, can we leverage on evaluation and relational database technology for query answering?

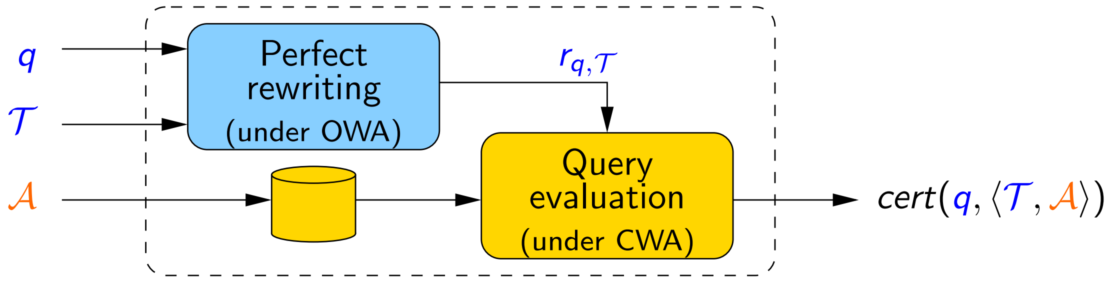
Outline

Compiling inference into evaluation for query answering

To be able to deal with data efficiently, we need to separate the contribution of $\mathcal{A}$ from the contribution of q and $\mathcal{T}$ and use evaluation.

$\leadsto$ Query answering by query rewriting.

Query rewriting

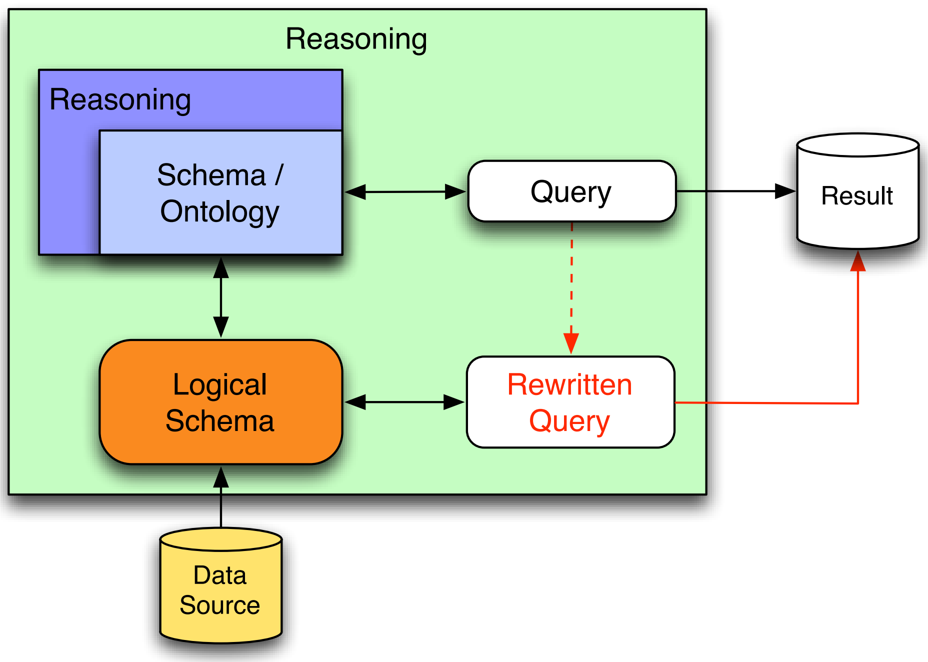


Query answering can always be thought as done in two phases:

1. Perfect rewriting: produce from q and the TBox $\mathcal{T}$ a new query $r_{q,\mathcal{T}}$ (called the perfect rewriting of q w.r.t. $\mathcal{T}$).
2. Query evaluation: evaluate $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$ seen as a complete database (and without considering the TBox $\mathcal{T}$).
   $\rightsquigarrow$ Produces $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$.

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

Query rewriting (cont'd)



Language of the rewriting

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

- When we can rewrite into FOL/SQL.
  $\leadsto$ Query evaluation can be done in SQL, i.e., via an RDBMS
  (*Note:* FOL is in LogSpace).
- When we can rewrite into an NLogSpace-hard language.
  $\leadsto$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\leadsto$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\leadsto$ Query evaluation requires (at least) power of Disjunctive Datalog.

Outline

Description Logics

In modeling an application domain we typically need to represent the domain of interest in terms of:

- objects
- classes
- relations (or associations)

and to reason about the representation

Description Logics (DLs) are logics specifically designed to represent and reason on:

- objects
- classes – called "concepts" in DLs
- (binary) relations – called "roles" in DLs

Brief history of DLs

Knowledge Representation is a subfield of Artificial Intelligence, see, e.g., [BCM+03].

- [late '70s, early '80s] – early days of KR formalisms
  - Semantic Networks: graph-based formalism, used to represent the meaning of sentences
  - Frame Systems: frames used to represent prototypical situations, antecedents of object-oriented formalisms

  Problems: no clear semantics, reasoning not well understood
- [mid '80s, '90s] – Description Logics (a.k.a. Concept Languages, Terminological Languages) developed starting in the mid '80s, with the aim of providing semantics and inference techniques to knowledge representation systems
- [Today] DLs are at the base of the whole research on ontology, and formalization of data conceptual modeling.

Current applications of DLs

DLs have evolved from being used "just" in KR

Found applications in:

- Databases:
  - schema design, schema evolution
  - query optimization
  - integration of heterogeneous data sources, data warehousing
- Conceptual modeling
- Foundation for the semantic web
- Ontology-Based Data Access (OBDA)
- ...

We will use to do query answering over UML class diagrams, which is related to OBDA [CDGL+05, CDGL+09, CDGL+13].

Note: To know more on DLs please take the course on Knowledge Representation and Semantic Technologies by R. Rosati (second semester).

The *DL-Lite* family

- A family of DLs optimized according to the tradeoff between expressive power and complexity of query answering, with emphasis on data [CDGL+05, CDGL+09, CDGL+13].
- Carefully designed to have nice computational properties for answering UCQs (i.e., computing certain answers):
  - The same complexity as relational databases.
  - In fact, query answering can be delegated to a relational DB engine.
  - The DLs of the *DL-Lite* family are essentially the maximally expressive ontology languages enjoying these nice computational properties.
- We present $DL\text{-}Lite_{\mathcal{A}}$, an expressive member of the *DL-Lite* family.

$DL\text{-}Lite_{\mathcal{A}}$ provides robust foundations for Ontology-Based Data Access.

DL-Lite$_{\mathcal{A}}$ KBs

TBox assertions:

- $C_1 \sqsubseteq C_2$ – class/"concept" inclusion assertions
- $C_1 \sqsubseteq \neg C_2$ – class /"concept" disjointness, aka "concept negative inclusion"

  where concepts are formed as: $C \longrightarrow A \mid \exists Q$

- $Q_1 \sqsubseteq Q_2$ – property /"role" inclusion assertions
- $Q_1 \sqsubseteq \neg Q_2$ – property /"role" disjointness, aka "role negative inclusion"

  where roles are formed as: $Q \longrightarrow P \mid P^-$

- (**funct** Q) – functionality assertions
- Proviso: functional properties cannot be specialized.

ABox assertions: $A(c)$, $P(c_1, c_2)$, with c_1, c_2 constants

Note: DL-Lite$_{\mathcal{A}}$ *distinguishes also between object and data properties (ignored here).*

Semantics of *DL-Lite*$_{\mathcal{A}}$

Construct	Syntax	Example	FOL translation
atomic conc.	A	Doctor	$A(x)$
atomic role	P	child	$P(x,y)$
exist. restr.	$\exists P$	∃child	$\exists y.P(x,y)$
	$\exists P^-$	∃child$^-$	$\exists y.P(y,x)$
conc. incl.	$C_1 \sqsubseteq C_2$	Father ⊑ ∃child	$\forall x.C_1(x) \rightarrow C_2(x)$
role incl.	$P_1 \sqsubseteq P_2$	hasFather ⊑ child	$\forall x,y.P_1(x,y) \rightarrow P_2(x,y)$
	$P_1 \sqsubseteq P_2^-$	hasFather ⊑ child$^-$	$\forall x,y.P_1(x,y) \rightarrow P_2(y,x)$
conc. disj.	$C_1 \sqsubseteq \neg C_2$	Kid ⊑ ¬∃child	$\forall x.C_1(x) \rightarrow \neg C_2(x)$
role disj.	$P_1 \sqsubseteq \neg P_2$	*(not part of UML)*	$\forall x,y.P_1(x,y) \rightarrow \neg P_2(x,y)$
	$P_1 \sqsubseteq \neg P_2^-$		$\forall x,y.P_1(x,y) \rightarrow \neg P_2(y,x)$
funct. asser.	(**funct** P)	(**funct** succ)	$\forall x,y,y'.P(x,y) \wedge P(x,y') \rightarrow y = y'$
	(**funct** P^-)	(**funct** succ$^-$)	$\forall x,y,y'.P(y,x) \wedge P(y',x) \rightarrow y = y'$
mem. asser.	$A(c)$	Father(bob)	$A(c)$
mem. asser.	$P(c_1,c_2)$	child(bob, ann)	$P(c_1,c_2)$

Note1: in database terms

- inclusion assertions ⇝ *inclusion dependencies* (a generalization of foreign keys)
- disjointness assertions ⇝ *disjointness constraints*
- functionality assertions ⇝ *functional dependencies* (a generalization of key constrains)
- membership assertions ⇝ *tuples on an incomplete database*

Note2: DL-Lite$_{\mathcal{A}}$ *adopts the Unique Name Assumption (UNA), i.e., different individuals denote different objects.*

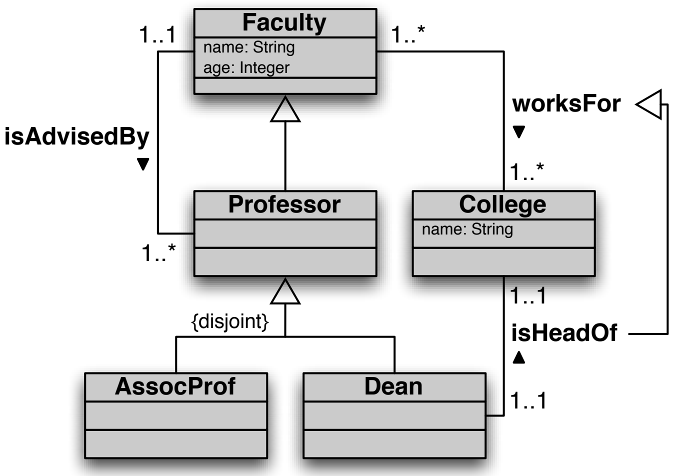
Capturing basic ontology constructs in *DL-Lite*$_\mathcal{A}$

ISA between classes	$A_1 \sqsubseteq A_2$
Disjointness between classes	$A_1 \sqsubseteq \neg A_2$
Domain and range of properties	$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$
Mandatory participation *(min card = 1)*	$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$
Functionality of relations *(max card = 1)*	(**funct** P) (**funct** P^-)
ISA between properties	$Q_1 \sqsubseteq Q_2$
Disjointness between properties	$Q_1 \sqsubseteq \neg Q_2$

Note without loosing its nice computational features:

- *DL-Lite*$_\mathcal{A}$ cannot capture completeness of a hierarchy. This would require disjunction (i.e., reasoning by cases).
- *DL-Lite*$_\mathcal{A}$ cannot capture subset constraints on association with max multiplicity different from "*". This may again introduce an hidden form of disjunction (i.e., reasoning by cases).
- *DL-Lite*$_\mathcal{A}$ can be extended to capture also min cardinality constraints "$A \sqsubseteq \leq nQ$" and max cardinality constraints "$A \sqsubseteq \geq nQ$".
- *DL-Lite*$_\mathcal{A}$ can be extended to capture also identification constraints "(**id** C $Q_1, \ldots, Q_n$)".

Translating UML Class Diagrams in *DL-Lite*$_\mathcal{A}$ KBs: example



$$
\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\\
\text{Faculty} & \sqsubseteq & \exists \mathit{age} \\
\exists \mathit{age}^- & \sqsubseteq & \texttt{xsd:integer} \\
& (\textbf{funct}\ \mathit{age}) & \\
\\
\exists\text{worksFor} & \sqsubseteq & \text{Faculty} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{College} \\
\text{Faculty} & \sqsubseteq & \exists\text{worksFor} \\
\text{College} & \sqsubseteq & \exists\text{worksFor}^- \\
\\
\exists\text{isHeadOf} & \sqsubseteq & \text{Dean} \\
\exists\text{isHeadOf}^- & \sqsubseteq & \text{College} \\
\text{Dean} & \sqsubseteq & \exists\text{isHeadOf} \\
\text{College} & \sqsubseteq & \exists\text{isHeadOf}^- \\
\text{isHeadOf} & \sqsubseteq & \text{worksFor} \\
& (\textbf{funct}\ \text{isHeadOf}) & \\
& (\textbf{funct}\ \text{isHeadOf}^-) & \\
& \vdots &
\end{array}
$$

Observations on *DL-Lite*$_{\mathcal{A}}$

- Captures all the basic constructs of UML Class Diagrams and of the ER Model ...
- ... except covering constraints in generalizations.
- Is the logical underpinning of OWL2 QL, one of the OWL 2 Profiles.
- Extends (the DL fragment of) the ontology language RDFS.
- Is completely symmetric w.r.t. direct and inverse properties.
- Does not enjoy the finite model property, i.e., reasoning and query answering differ depending on whether we consider or not also infinite models.

Technical properties of *DL-Lite*$_{\mathcal{A}}$

- Completely symmetric w.r.t. direct and inverse roles: roles are always navigable in the two directions
- TBoxes may contain cyclic dependencies (which typically increase the computational complexity of reasoning)

  Example: $A \sqsubseteq \exists P, \quad \exists P^- \sqsubseteq A$
- Does not enjoy the finite model property, unless we drop functional assertions.

Technical properties of *DL-Lite*: no finite model property

DL-Lite does not enjoy the finite model property.

Example

TBox $\mathcal{T}$: $\mathsf{Nat} \sqsubseteq \exists\mathsf{succ}$ $\quad \exists\mathsf{succ}^- \sqsubseteq \mathsf{Nat}$

$\mathsf{Zero} \sqsubseteq \mathsf{Nat}$ $\quad \mathsf{Zero} \sqsubseteq \neg\exists\mathsf{succ}^-$ $\quad (\textbf{funct}\ \mathsf{succ}^-)$

ABox $\mathcal{A}$: $\mathsf{Zero}(0)$

$\mathcal{K} = \langle\mathcal{T},\mathcal{A}\rangle$ admits only infinite models.
Hence, it is satisfiable, but not finitely satisfiable.

Hence, reasoning w.r.t. arbitrary models is different from reasoning w.r.t. finite models only.

Query answering in $DL\text{-}Lite_{\mathcal{A}}$

- We study query answering via query rewriting for UCQs over $DL\text{-}Lite_{\mathcal{A}}$ KBs/ontologies.
- We focus on query answering over satisfiable KBs, i.e., KBs that admit at least one model.
- We show how to exploit query answering over satisfiable KBs to establish KB satisfiability itself.
- We show how to reduce the other usual intensional reasoning tasks to KB satisfiability checking.

Remark

we call positive inclusions (PIs) assertions of the form

$$C_1 \sqsubseteq C_2$$
$$Q_1 \sqsubseteq Q_2$$

whereas we call negative inclusions (NIs) assertions of the form

$$C_1 \sqsubseteq \neg C_2$$
$$Q_1 \sqsubseteq \neg Q_2$$

Query answering over satisfiable *DL-Lite*$_\mathcal{A}$ KBs

Theorem

Let q be a boolean UCQs and $\mathcal{T} = \mathcal{T}_{\rm PI} \cup \mathcal{T}_{\rm NI} \cup \mathcal{T}_{\sf funct}$ be a TBox s.t.

- $\mathcal{T}_{\rm PI}$ is a set of PIs
- $\mathcal{T}_{\rm NI}$ is a set of NIs
- $\mathcal{T}_{\sf funct}$ is a set of functionalities.

For each ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, we have that

$$\langle \mathcal{T}, \mathcal{A} \rangle \models q \text{ iff } \langle \mathcal{T}_{\rm PI}, \mathcal{A} \rangle \models q.$$

Proof [intuition]

q is a positive query, i.e., it does not contain atoms with negation nor inequality. $\mathcal{T}_{\rm NI}$ and $\mathcal{T}_{\sf funct}$ only contribute to infer new negative consequences, i.e, sentences involving negation.

If q is non-boolean, we have that $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(q, \langle \mathcal{T}_{\rm PI}, \mathcal{A} \rangle)$.

Satisfiability of *DL-Lite*$_\mathcal{A}$ KBs

$\langle \mathcal{T}, \emptyset \rangle$ is always satisfiable. Indeed, always admits the model where the extension of all concepts and roles is empty. Hence, inconsistency in *DL-Lite*$_\mathcal{A}$ may arise only when ABox assertions contradict the TBox.

$\langle \mathcal{T}_{\rm PI}, \mathcal{A} \rangle$, where $\mathcal{T}_{\rm PI}$ contains only PIs, is always satisfiable. Indeed, always admits the model where extension of concepts and roles being the total relations of arity 1 and 2 over the interpretation domain. Hence, inconsistency in *DL-Lite*$_\mathcal{A}$ may arise only when ABox assertions violate functionalities or NIs.

Only when we have both functionalities and of NIs in the TBox and a non-empty ABox that satisfiability becomes an issue.

Example: **TBox** $\mathcal{T}$**:** Professor $\sqsubseteq \neg$Student
$\exists$teaches $\sqsubseteq$ Professor
(**funct** teaches$^-$)

ABox $\mathcal{A}$**:** teaches($\texttt{John}, \texttt{databases}$)
Student($\texttt{John}$)
teaches($\texttt{Mark}, \texttt{databases}$)

Interestingly, violations of functionalities and of NIs can be checked separately!

Satisfiability of *DL-Lite*$_\mathcal{A}$ KBs: checking functs

Theorem

Let $\mathcal{T}_{\rm PI}$ be a TBox with only PIs, and (**funct** Q) a functionality assertion. Then, for any ABox $\mathcal{A}$,
$\langle \mathcal{T}_{\rm PI} \cup \{(\textbf{funct } Q)\}, \mathcal{A}\rangle$ is sat iff $\mathcal{A} \not\models \exists x, y, z. Q(x, y) \wedge Q(x, z) \wedge y \neq z$. *Note in the latter $\mathcal{A}$ is considered as a complete database!*

Proof [sketch]

$\langle \mathcal{T}_{\rm PI} \cup \{(\textbf{funct } Q)\}, \mathcal{A}\rangle$ is satisfiable iff $\langle \mathcal{T}_{\rm PI}, \mathcal{A}\rangle \not\models \neg(\textbf{funct } Q)$. This holds iff $\mathcal{A} \not\models \neg(\textbf{funct } Q)$ (separability property – sophisticated proof). From separability, the claim easily follows, by noticing that (**funct** Q) corresponds to the FOL sentence $\forall x, y, z. Q(x, y) \wedge Q(x, z) \rightarrow y = z$.

For a set of functionalities, we take the union of sentences of the form above (which corresponds to a boolean FOL query).

Checking satisfiability wrt functionalities therefore amounts to evaluate a FOL query over the ABox.

Checking functs: example

TBox $\mathcal{T}$: Professor $\sqsubseteq \neg$Student
$\exists$teaches $\sqsubseteq$ Professor
(**funct** teaches$^-$)

The query we associate to the functionality is:

$$q() \leftarrow \text{teaches}(y, x), \text{teaches}(z, x), y \neq z$$

which evaluated over the ABox

ABox $\mathcal{A}$: teaches(John, databases)
Student(John)
teaches(Mark, databases)

returns true.

Satisfiability of *DL-Lite*$_\mathcal{A}$ KBs: checking NIs

Theorem

Let $\mathcal{T}_{\mathrm{PI}}$ be a TBox with only PIs, and $A_1 \sqsubseteq \neg A_2$ a NI. For any ABox $\mathcal{A}$, $\langle \mathcal{T}_{\mathrm{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ is sat iff $\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A}\rangle \not\models \exists x.A_1(x) \wedge A_2(x)$.

Proof [sketch]

$\langle \mathcal{T}_{\mathrm{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ is satisfiable iff $\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A}\rangle \not\models \neg(A_1 \sqsubseteq \neg A_2)$. The claim follows easily by noticing that $A_1 \sqsubseteq \neg A_2$ corresponds to the FOL sentence $\forall x.A_1(x) \rightarrow \neg A_2(x)$.

The property holds for all kinds of NIs ($A \sqsubseteq \exists Q$, $\exists Q_1 \sqsubseteq \exists Q_2$, etc.)

For a set of NIs, we take the union of sentences of the form above (which corresponds to a UCQ).

Checking satisfiability wrt NIs amounts to answering a UCQ over a KB with only PIs (this can be reduced to evaluating a UCQ over the ABox – see later).

Checking NIs: example

TBox $\mathcal{T}$**:** Professor $\sqsubseteq$ ¬Student
$\exists$teaches $\sqsubseteq$ Professor
(**funct** teaches$^-$)

The query we associate to the NI is:

$$q() \leftarrow \text{Student}(x), \text{Professor}(x)$$

whose answer over the KB $\mathcal{K}_{pi}$ formed by PIs only and ABox:

$\mathcal{K}_{pi}$**:** $\exists$teaches $\sqsubseteq$ Professor
teaches(`John`, `databases`)
Student(`John`)
teaches(`Mark`, `databases`)

is true.

Example

Example: **PI** $\mathcal{T}_P$: $\exists$teaches $\sqsubseteq$ Professor

NI N: Professor $\sqsubseteq \neg$Student

Query q_N: $q() \leftarrow \text{Student}(x), \text{Professor}(x)$

Perfect Rewriting $r_{q,\mathcal{T}_P}$: $q() \leftarrow \text{Student}(x), \text{Professor}(x)$
$q() \leftarrow \text{Student}(x), \text{teaches}(x, y)$

ABox $\mathcal{A}$: teaches(John, databases)
Student(John)

It is easy to see that $r_{q,\mathcal{T}_P}$ evaluates to *true* over $\mathcal{A}$, and that therefore $\mathcal{K}$ is unsatisfiable.

Checking satisfiability of *DL-Lite*$_\mathcal{A}$ KBs

Checking satisfiability

Satisfiability of a *DL-Lite*$_\mathcal{A}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluation of a first order query over $\mathcal{A}$, obtained by uniting

(a) the FOL query associated to functionalities in $\mathcal{T}$ to

(b) the UCQs produced by a rewriting procedure (depending only on the PIs in $\mathcal{T}$) applied to the query associated to NIs in $\mathcal{T}$.

$\rightsquigarrow$ KB satisfiability in *DL-Lite*$_\mathcal{A}$ can be done using RDMBS technology.

Other intensional tasks of *DL-Lite*$_\mathcal{A}$ KBs

All other intensional reasoning tasks,such as class consistency, logical implication, etc., can all be reduced to KB satisfiability.

Checking intensional tasks

- Class consistency: to check $\mathcal{T} \not\models C_1 \sqsubseteq \mathit{false}$, check satisfiability of
$$\mathcal{K} = \langle \mathcal{T} \cup \{A_{new} \sqsubseteq C\}, \{A_{new}(c_{new})\}\rangle$$
- Logical implication of PI concept inclusions: to check $\mathcal{T} \models C_1 \sqsubseteq C_2$, check unsatisfiability of
$$\mathcal{K} = \langle \mathcal{T} \cup \{A_{new} \sqsubseteq C_1, A_{new} \sqsubseteq \neg C_2\}, \{A_{new}(c_{new})\}\rangle$$
- Logical implication of NI concept inclusions: to check $\mathcal{T} \models C_1 \sqsubseteq \neg C_2$, check unsatisfiability of
$$\mathcal{K} = \langle \mathcal{T} \cup \{A_{new} \sqsubseteq C_1, A_{new} \sqsubseteq C_2\}, \{A_{new}(c_{new})\}\rangle$$
- Logical implication of PI role inclusions: to check $\mathcal{T} \models Q_1 \sqsubseteq Q_2$, check unsatisfiability of
$$\mathcal{K} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq Q_1, P_{new} \sqsubseteq \neg Q_2\}, \{A_{new}(c_{new}, c'_{new})\}\rangle$$
- Logical implication of NI role inclusions: to check $\mathcal{T} \models Q_1 \sqsubseteq \neg Q_2$, check unsatisfiability of
$$\mathcal{K} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq Q_1, P_{new} \sqsubseteq Q_2\}, \{A_{new}(c_{new}, c'_{new})\}\rangle$$
- Logical implication of functional assertions: $\mathcal{T} \models$ (**funct** Q), trivial: always false!!!

(A_{new} isa a new concept, P_{new} a new role, and c_{new}, c'_{new} new constants.)

Query answering in *DL-Lite*$_\mathcal{A}$: query rewriting

To the aim of answering queries, from now on we assume that $\mathcal{T}$ contains only PIs.

Given a CQ q and a satisfiable KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$, we compute $cert(q, \mathcal{K})$ as follows

1. using $\mathcal{T}$, reformulate q as a union $r_{q,\mathcal{T}}$ of CQs.
2. Evaluate $r_{q,\mathcal{T}}$ directly over $\mathcal{A}$ managed in secondary storage via a RDBMS.

Correctness of this procedure shows FOL-rewritability of query answering in *DL-Lite*$_\mathcal{A}$
$\leadsto$ Query answering over *DL-Lite*$_\mathcal{A}$ KBs can be done using RDMBS technology.

Query answering in *DL-Lite*$_{\mathcal{A}}$: query rewriting

Expansion step:

- **when** an atom of the query unifies with the **right-hand-side** of a PI (with substitution σ).
- **substitute** the atom with the **left-hand-side** of the PI (expressed in FOL, and to which σ is applied).
- **add** the resulting query to the UCQ to return.

The basic case:

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

$$\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$$

as a logic rule: $\mathsf{Professor}(z) \leftarrow \mathsf{AssProfessor}(z)$

Towards the computation of the perfect rewriting, we add to the input query above the following query ($\sigma = \{z/x\}$)

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

We say that the PI $\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$ applies to the atom $\mathsf{Professor}(x)$.

Query answering in *DL-Lite*$_{\mathcal{A}}$: query rewriting

Consider now the query

$$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y)$$

$$\mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$$

as a logic rule: $\mathsf{teaches}(z_1, z_2) \leftarrow \mathsf{Professor}(z_1)$

We add to the reformulation the query ($\sigma = \{z_1/x, z_2/y\}$)

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

Query answering in *DL-Lite*$_{\mathcal{A}}$: query rewriting

Conversely, for the query

$$q(x) \leftarrow \text{teaches}(x, \texttt{databases})$$

$$\text{Professor} \sqsubseteq \exists\text{teaches}$$
$$\text{as a logic rule:} \quad \text{teaches}(z_1, z_2) \leftarrow \text{Professor}(z_1)$$

$\text{teaches}(x, \texttt{databases})$ does not unify with $\text{teaches}(z_1, z_2)$, since the existentially quantified variable z_2 in the head of the rule does not unify with the constant $\texttt{databases}$.

In this case the PI does not apply to the atom $\text{teaches}(x, \texttt{databases})$.

The same holds for the following query, where y is distinguished

$$q(x, y) \leftarrow \text{teaches}(x, y)$$

Query answering in *DL-Lite*$_{\mathcal{A}}$: query rewriting

An analogous behavior with join variables

$$q(x) \leftarrow \text{teaches}(x, y), \text{Course}(y)$$

$$\text{Professor} \sqsubseteq \exists\text{teaches}$$
$$\text{as a logic rule:} \quad \text{teaches}(z_1, z_2) \leftarrow \text{Professor}(z_1)$$

The PI above does not apply to the atom $\text{teaches}(x, y)$.

Conversely, the PI

$$\exists\text{teaches}^- \sqsubseteq \text{Course}$$
$$\text{as a logic rule:} \quad \text{Course}(z_2) \leftarrow \text{teaches}(z_1, z_2)$$

applies to the atom $\text{Course}(y)$.

We add to the perfect rewriting the query ($\sigma = \{z_2/y\}$)

$$q(x) \leftarrow \text{teaches}(x, y), \text{teaches}(z_1, y)$$

Query answering in $DL\text{-}Lite_{\mathcal{A}}$: query rewriting

Unification Step (aka called "reduce")

when two atoms of the query unify with substitution σ.

unify by applying substitution σ to all atoms, and remove duplicate atoms from the resulting query.

add add the resulting query to the UCQ to return.

Consider the query

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

The PI $\quad$ Professor $\sqsubseteq \exists$teaches

as a logic rule: $teaches(z_1, z_2) \leftarrow Professor(z_1)$

does not apply to $teaches(x, y)$ nor $teaches(z, y)$, since y is a join variable.

However, we can transform the above query by unifying the atoms $teaches(x, y)$, $teaches(z_1, y)$.
The unification step produces ($\sigma = \{z_1/x, z_2/y\}$) the following query

$$q(x) \leftarrow teaches(x, y)$$

We can now apply the PI above and add to the reformulation the query

$$q(x) \leftarrow Professor(x)$$

Answering by rewriting in $DL\text{-}Lite_{\mathcal{A}}$: algorithm

Query Rewriting Algorithm (naive version)

Given the (U)CQ q over a $DL\text{-}Lite_{\mathcal{A}}$ TBox $\mathcal{T}$ generate a UCQ q_r by

- Include the original query q itself in q_r.
- Apply q in all possible ways the expansion steps and unification steps in all possible way, adding the results which are CQs to q_r.
- Stop when expansion steps and unification steps do not add new CQs to q_r.

Theorem

The UCQ q_r resulting from this process is the perfect rewriting of q over $\mathcal{T}$, in the sense that for every ABox $\mathcal{A}$ we have:

$$cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = q_r^{\mathcal{A}}$$

That is: to compute the certain answer of the (U)CQ q over the KB $\langle \mathcal{T}, \mathcal{A} \rangle$ evaluate the UCQ $r_{q,\mathcal{T}}$ over $\mathcal{A}$ seen as a DB.

Query answering in $DL\text{-}Lite_{\mathcal{A}}$: example

TBox: $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$
$\exists \mathsf{teaches}^- \sqsubseteq \mathsf{Course}$

Query: $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,y), \mathsf{Course}(y)$

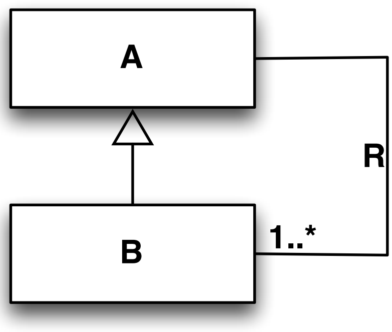
Perfect Rewriting: $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,y), \mathsf{Course}(y)$
$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,y), \mathsf{teaches}(z,y)$
$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,y)$
$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$

ABox: $\mathsf{teaches}(\mathtt{John}, \mathtt{databases})$
$\mathsf{Professor}(\mathtt{Mary})$

It is easy to see that the evaluation of $r_{q,\mathcal{T}}$ over $\mathcal{A}$ in this case produces the set $\{\mathtt{John}, \mathtt{Mary}\}$.

Exercise

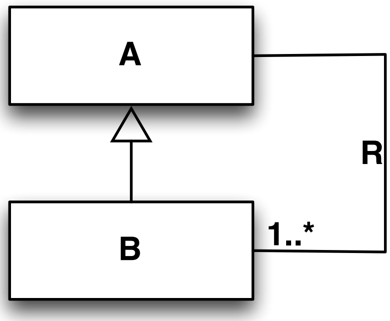
Express in $DL\text{-}Lite_{\mathcal{A}}$ the following ontology:



Considering the following ABox $\mathcal{A} = \{B(c)\}$ compute the answer to the following query:

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

Exercise (solution)



TBox: $B \sqsubseteq A$
$\exists R. \sqsubseteq A$
$\exists R^-. \sqsubseteq B$
$A \sqsubseteq \exists R.$

ABox: $B(c)$

Expansions:

$q(x)$	$\leftarrow$	$R(x,y), R(y,z).$	
$q(x)$	$\leftarrow$	$R(x,y), A(y).$	expanded using $A \sqsubseteq \exists R$ (note: z isolated)
$q(x)$	$\leftarrow$	$R(x,y), B(y).$	expanded using $B \sqsubseteq A$
$q(x)$	$\leftarrow$	$R(x,y), R(w,y).$	expanded using $\exists R^- \sqsubseteq B$
$q(x)$	$\leftarrow$	$R(x,y).$	unified: $w = x$
$q(x)$	$\leftarrow$	$A(x).$	expanded using $A \sqsubseteq \exists R$ (note: y isolated)
$q(x)$	$\leftarrow$	$B(x).$	expanded using $B \sqsubseteq A$
			$\Longrightarrow$ answer $x = c$

Query answering in $DL\text{-}Lite_{\mathcal{A}}$: another example

TBox: $\text{Person} \sqsubseteq \exists\text{hasFather}$
$\exists\text{hasFather}^- \sqsubseteq \text{Person}$

ABox: $\text{Person}(\texttt{Mary})$

Query: $\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, y_3)$

$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, \_)$

$\Downarrow$ Apply $\text{Person} \sqsubseteq \exists\text{hasFather}$ to the atom $\text{hasFather}(y_2, \_)$

$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{Person}(y_2)$

$\Downarrow$ Apply $\exists\text{hasFather}^- \sqsubseteq \text{Person}$ to the atom $\text{Person}(y_2)$

$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(\_, y_2)$

$\Downarrow$ Unify atoms $\text{hasFather}(y_1, y_2)$ and $\text{hasFather}(\_, y_2)$

$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2)$

$\Downarrow$

...

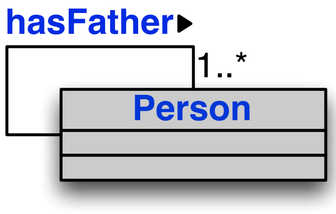
$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, \_)$

$\Downarrow$ Apply $\text{Person} \sqsubseteq \exists\text{hasFather}$ to the atom $\text{hasFather}(x, \_)$

$\text{q}(x) \leftarrow \text{Person}(x)$

Query answering in $DL\text{-}Lite_{\mathcal{A}}$: exercise

Consider the following example, seen before. Compute certain answers through rewriting.



TBox: $\exists$hasFather $\sqsubseteq$ Person
$\exists$hasFather$^-$ $\sqsubseteq$ Person
Person $\sqsubseteq$ $\exists$hasFather

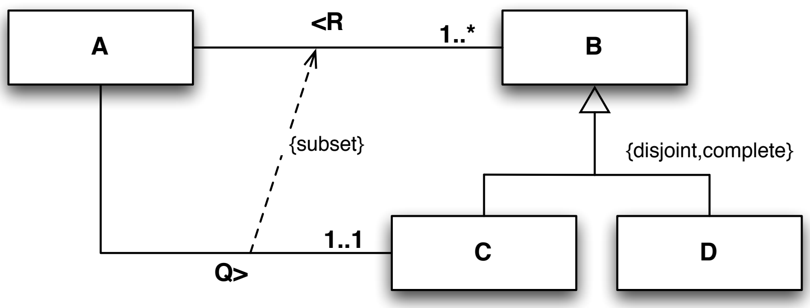
ABox: Person(john)
Person(paul)
Person(toni)
hasFather(john,paul)
hasFather(paul,toni)

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

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

Exercise 1

Express in $DL\text{-}Lite_{\mathcal{A}}$ the following ontology:



Considering the following ABox $\mathcal{A} = \{A(a)\}$ compute the answer to the following queries:

$$q(x) \leftarrow Q(x,y), R(y,z).$$
$$q'() \leftarrow B(x).$$

Exercise 1 (solution)

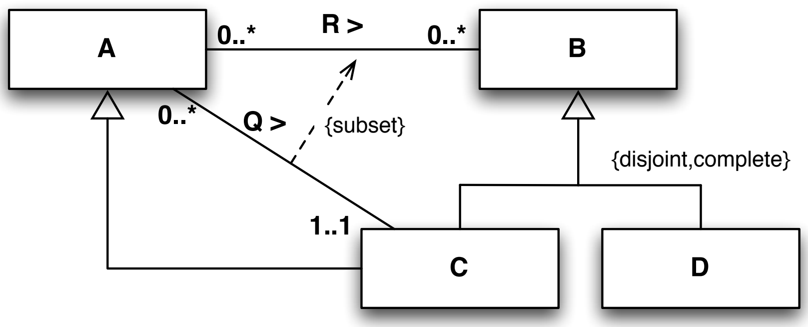
Expansions:

$$
\begin{array}{lcll}
q(x) & \leftarrow & Q(x,y), R(y,z). & \\
q(x) & \leftarrow & Q(x,y), Q(z,y). & Q \sqsubseteq R^- \\
q(x) & \leftarrow & Q(x,y). & \text{unify: } z = x \\
q(x) & \leftarrow & A(x). & A \sqsubseteq \exists Q \\
 & & & \Longrightarrow \text{ answer } x = a
\end{array}
$$

$$
\begin{array}{lcll}
q'() & \leftarrow & B(x). & \\
q'() & \leftarrow & R(x,y). & \exists R. \sqsubseteq B \\
q'() & \leftarrow & A(y). & A \sqsubseteq \exists R^- \\
 & & & \Longrightarrow \text{ answer } true \text{ (by } y = a)
\end{array}
$$

Exercise 2

Express in $DL\text{-}Lite_{\mathcal{A}}$ the following ontology:



Considering the following ABox $\mathcal{A} = \{Q(a,b), R(b,b), C(c)\}$ compute the answer to the following queries:

$$q(x) \leftarrow R(x,y), R(y,z), A(z).$$

Exercise 2 (solution)

Expansions:

```
q(x) :- R(x,y), R(y,z), A(z).
q(x) :- R(x,x), A(x).       --- unify
q(x) :- R(x,x), R(x,y).     --- Exists R ISA A
q(x) :- R(x,x).             --- unify

  answer x = b

......
```

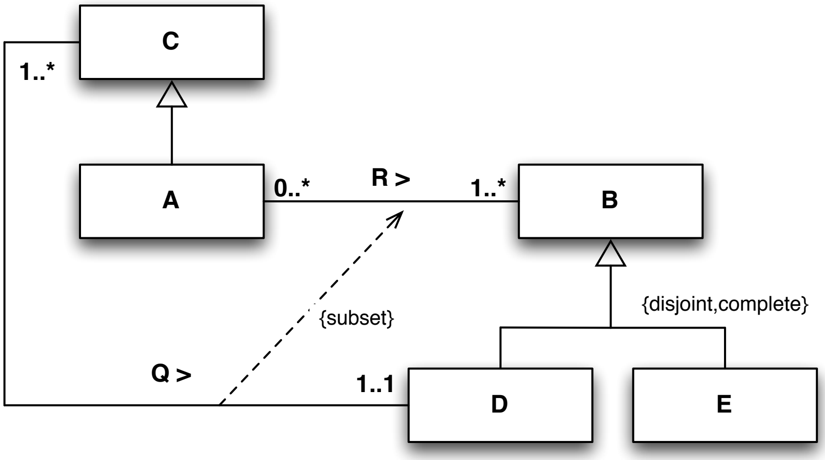

Exercise 2 (solution)

Expansions:

```
.....

q(x) :- R(x,y), R(y,z), A(z).
q(x) :- R(x,y), R(y,z), C(z).   --- C ISA A
q(x) :- R(x,y), R(y,z), Q(w,z). --- Exists Q- ISA C
q(x) :- R(x,y), Q(y,z), Q(w,z). --- Q ISA R
q(x) :- R(x,y), Q(y,z).         --- unify
q(x) :- R(x,y), A(y).           --- A ISA Exists Q
q(x) :- R(x,y), C(y).           --- C ISA A
q(x) :- R(x,y), Q(z,y).         --- Exists Q- ISA C
q(x) :- Q(x,y), Q(z,y).         --- Q ISA R
q(x) :- Q(x,y).                 --- unify

   answer x = a

q(x) :- A(x).                   --- A ISA Exists Q
q(x) :- C(x).                   --- C ISA A

   answer x = c
```

Exercise 3

Express in *DL-Lite*$_{\mathcal{A}}$ the following ontology:



Considering the following ABox $\mathcal{A} = \{C(a)\}$ compute the answer to the following queries:

$$q(x) \leftarrow R(x,y), B(y).$$
$$q'(x) \leftarrow A(x).$$

Can we simplify the diagram?

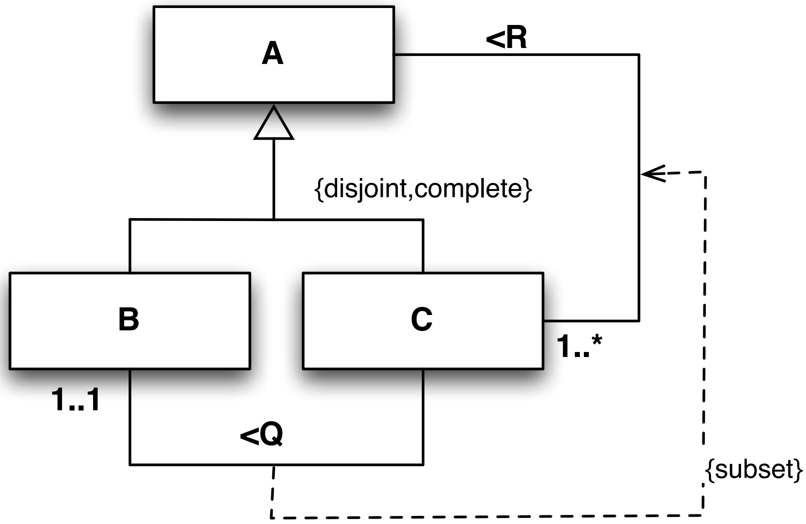
Exercise 3 (solution)

Expansions:

```
q(x) :- R(x,y), B(y).
q(x) :- R(x,y), D(y).    --- D ISA B
q(x) :- R(x,y), Q(z,y). --- Exists Q- ISA D
q(x) :- Q(x,y), Q(z,y). --- Q ISA R
q(x) :- Q(x,y).          --- unify
q(x) :- C(x).            --- C ISA Exists Q

   answer x = a


q'(x):- A(x).
q'(x):- R(x,y).   --- A ISA Exists R
q'(x):- Q(x,y).   --- Q ISA R
q'(x):- C(x).     --- C ISA Exists Q

   answer x = a
```

Exercise 4

Express in $\textit{DL-Lite}_{\mathcal{A}}$ the following ontology:



Considering the following ABox $\mathcal{A} = \{B(b)\}$ compute the answer to the following queries:

$$q(z) \leftarrow R(x,y), R(y,z).$$
$$q'() \leftarrow C(x).$$

Exercise 4 (solution)

Expansions:

```
q(z) :- R(x,y), R(y,z).
q(z) :- A(y), R(y,z).    --- A ISA Exists R-
q(z) :- C(y), R(y,z).    --- C ISA A
q(z) :- R(y,w), R(y,z). --- Exists R ISA C
q(z) :- R(y,z).          --- unify
q(z) :- A(z).            --- A ISA Exists R-
q(z) :- B(z).            --- B ISA A

   answer z = b


q'() :- C(x).
q'() :- R(x,y).    -- Exists R ISA C
q'() :- A(y).      -- A ISA Exists R-
q'() :- B(y).      -- B ISA A

   answer z = b
```

Complexity of reasoning in $DL\text{-}Lite_{\mathcal{A}}$

KB satisfiability and all classical DL reasoning tasks are:

- Efficiently tractable in the size of TBox (i.e., PTIME).
- Very efficiently tractable in the size of the ABox (i.e., LOGSPACE).

In fact, reasoning can be done by constructing suitable FOL/SQL queries and evaluating them over the ABox (FOL-rewritability).

Query answering for CQs and UCQs is:

- PTIME in the size of TBox.
- LOGSPACE in the size of the ABox.
- Exponential in the size of the query (NP-complete).
  Bad? ...not really, this is exactly as in relational DBs.

Can we go beyond $DL\text{-}Lite_{\mathcal{A}}$?

By adding essentially any other DL construct, e.g., union ($\sqcup$), value restriction ($\forall R.C$), etc., without some limitations we lose these nice computational properties (see later).

Beyond $DL\text{-}Lite_{\mathcal{A}}$: results on data complexity

	lhs	rhs	funct.	Prop. incl.	Data complexity of query answering
0	$DL\text{-}Lite_{\mathcal{A}}$		$\surd^*$	$\surd^*$	in LOGSPACE
1	$A \mid \exists P.A$	A	–	–	NLOGSPACE-hard
2	A	$A \mid \forall P.A$	–	–	NLOGSPACE-hard
3	A	$A \mid \exists P.A$	$\surd$	–	NLOGSPACE-hard
4	$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	–	–	PTIME-hard
5	$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	–	–	PTIME-hard
6	$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	$\surd$	–	PTIME-hard
7	$A \mid \exists P.A \mid \exists P^-.A$	$A \mid \exists P$	–	–	PTIME-hard
8	$A \mid \exists P \mid \exists P^-$	$A \mid \exists P \mid \exists P^-$	$\surd$	$\surd$	PTIME-hard
9	$A \mid \neg A$	A	–	–	coNP-hard
10	A	$A \mid A_1 \sqcup A_2$	–	–	coNP-hard
11	$A \mid \forall P.A$	A	–	–	coNP-hard

Notes:

- * with the "proviso" of not specializing functional properties.
- NLOGSPACE and PTIME hardness holds already for instance checking.
- For coNP-hardness in line 10, a TBox with a single assertion $A_L \sqsubseteq A_T \sqcup A_F$ suffices! $\leadsto$ No hope of including covering constraints.

Beyond union of conjunctive queries

Till now we have assumed that the client queries are UCQs (aka positive queries). Can we go beyond UCQ? Can we go to full FOL/SQL queries?

- No! Answering FOL queries in presence of incomplete information is undecidable: Consider an empty source (no data), still a (boolean) FOL query may return *true* because it is valid! (FOL validity is undecidable)
- Yes! With some compromises:
  Query what the ontology knows about the domain, not what is true in the domain!
  On knowledge we have complete information, so evaluating FOL queries is LogSpace.

SparSQL

Full SQL, but with relations in the FROM clause that are UCQs, expressed in SPARQL, over the ontology.

- SPARQL queries are used to query what is true in the domain.
- SQL is used to query what the ontology knows about the domain.

Example: negation

Return all known people that are neither known to be male nor known to be female.

```
SELECT persons.x
FROM SparqlTable(SELECT ?x
                 WHERE {?x rdf:type 'Person'}
                ) persons
EXCEPT  (
SELECT males.x
FROM SparqlTable(SELECT ?x
                 WHERE {?x rdf:type 'Male'}
                ) males
UNION
SELECT females.x
FROM SparqlTable(SELECT ?x
                 WHERE {?x rdf:type 'Female'}
                ) females
)
```

Example: aggregates

Return the people and the number of their known spouses, but only if they are known to be married to at least two people.

```
SELECT marriage.x, count(marriage.y)
FROM SparqlTable(SELECT ?x ?y
                 WHERE {?x :MarriedTo ?y}
                ) marriage
GROUP BY marriage.x
HAVING count(marriage.y) >= 2
```

SparSQL in *DL-Lite*$_{\mathcal{A}}$

Answering of SparSQL queries in *DL-Lite*$_{\mathcal{A}}$:

1. Expand and unfold the UCQs (in the SparqlTables) as usual in *DL-Lite*$_{\mathcal{A}}$ $\leadsto$ an SQL query over the ABox (seen as a database) for each SparqlTable in the FROM clauses.
2. Substitute SparqlTables with the new SQL queries. $\leadsto$ the result is again an SQL query over the ABox (seen as a database)!
3. Evaluate the resulting SQL query over the ABox (seen as a database)

Outline

References

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation and Applications.
Cambridge University Press, 2003.

[CDGL+05] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
DL-Lite: Tractable description logics for ontologies.
In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[CDGL+09] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati.
Ontologies and databases: The dl-lite approach.
In *Semantic Technologies for Informations Systems - 5th Int. Reasoning Web Summer School (RW 2009)*, volume 5689 of *Lecture Notes in Computer Science*, pages 255–356. Springer, 2009.

[CDGL+13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Data complexity of query answering in description logics.
Artificial Intelligence, 195:335–360, 2013.

[IL84] T. Imielinski and W. J. Lipski.
Incomplete information in relational databases.
J. of the ACM, 31(4):761–791, 1984.