Ontology Based Data Access
Vision: Ontologies at the Core of Information Systems

• Usage of all system resources (data and services) is done through a domain conceptualization.

• Cooperation between systems is done at the level of the conceptualizations.

• This implies:
  
  – Hide to the user where and how data and services are stored or implemented;
  
  – Present to the user a conceptual view of the data and services.
Ontology based Data Access

- An ontology provides meta-information about the data and the vocabulary used to query the data. It can impose constraints on the data.

- Actual data can be incomplete w.r.t. such meta-information and constraints. So data should be stored using open world semantics rather than closed world semantics: use ABoxes instead of relational database instances.

- During query answering, the system has to take into account the ontology.

We discuss ontology based data access in the framework of description logic knowledge bases.
Knowledge Base (KB)

**TBox** (terminological box, schema)

- \( \text{Man} \equiv \text{Human} \cap \text{Male} \)
- \( \text{Father} \equiv \text{Man} \cap \exists \text{hasChild} \)
- ...

**ABox** (assertion box, data)

- john : Man
- (john, mary) : hasChild
- ...

Inference System

Interface
Knowledge Base (= Ontology with database instance)

A knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and a simple ABox $\mathcal{A}$ (or, equivalently, a database instance).

We combine the open world semantics for TBoxes and ABoxes in the obvious manner, and obtain an open world semantics for knowledge bases.

An interpretation $\mathcal{I}$ satisfies a knowledge base $(\mathcal{T}, \mathcal{A})$, in symbols

$$\mathcal{I} \models (\mathcal{T}, \mathcal{A}),$$

if it satisfies both $\mathcal{T}$ and $\mathcal{A}$. In this case we also say that $\mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{A})$. The set of models of $(\mathcal{T}, \mathcal{A})$ is denoted by $\text{Mod}(\mathcal{T}, \mathcal{A})$. 
Certain Answers

Given a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and an FOPL query $F(x_1, \ldots, x_k)$, we say that $(a_1, \ldots, a_k)$ is a certain answer to $F(x_1, \ldots, x_k)$ by $\mathcal{K}$, in symbols

$$\mathcal{K} \models F(a_1, \ldots, a_k),$$

if

- $a_1, \ldots, a_k$ are individual names in $\mathcal{A}$;
- for all interpretations $\mathcal{I}$:

$$\mathcal{I} \models \mathcal{K} \Rightarrow \mathcal{I} \models F(a_1, \ldots, a_k).$$

The set of certain answers given to $F$ by $\mathcal{K}$ is defined as:

$$\text{certanswer}(F, \mathcal{K}) = \{(a_1, \ldots, a_k) \mid \mathcal{K} \models F(a_1, \ldots, a_k)\}$$
Let $\mathcal{K}$ be a knowledge base. For a query $F$ without variables (Boolean query), we say that

- the certain answer given by $\mathcal{K}$ is “yes” if $\mathcal{I} \models F$, for all interpretations $\mathcal{I}$ satisfying $\mathcal{K}$;

- the certain answer given by $\mathcal{K}$ is “no” if $\mathcal{I} \not\models F$, for all interpretations $\mathcal{I}$ satisfying $\mathcal{K}$.

- Otherwise the certain answer is: “Don’t know”.
Example

Consider the TBox $\mathcal{T}_U$:

- BritishUniversity ⊑ University;
- University $\sqcap$ Student ⊑ ⊥;
- $\top \sqsubseteq \forall \text{registered at University};$
- $\top \sqsubseteq \forall \text{student at University};$
- $\exists \text{student at } \top \sqsubseteq \text{Student};$
- Student ⊑ $\exists \text{student at } \top;$
- NonBritishUni ≡ University $\sqcap \neg$BritishUniversity.
and the simple ABox (equivalently, database instance) \( \mathcal{A} \):

- NonBritishUni(CMU)
- Institution(Harvard), Institution(FUBerlin)
- BritishUniversity(LU), BritishUniversity(MU)
- Student(Tim)
- registered(Tim, LU), registered(Bob, MU)
- student_at(Tom, Harvard)
Example (continued)

Denote by $\mathcal{I}_\mathcal{A}$ the interpretation corresponding to the database instance $\mathcal{A}$:

- $\Delta_{\mathcal{I}_\mathcal{A}} = \{\text{CMU, Harvard, FUBerlin, Tim, Tom, Bob, MU, LU}\}$;
- $\text{NonBritishUni}_{\mathcal{I}_\mathcal{A}} = \{\text{CMU}\}$;
- $\text{Institution}_{\mathcal{I}_\mathcal{A}} = \{\text{Harvard, FUBerlin}\}$;
- $\text{BritishUniversity}_{\mathcal{I}_\mathcal{A}} = \{\text{LU, MU}\}$;
- $\text{Student}_{\mathcal{I}_\mathcal{A}} = \{\text{Tim}\}$;
- $\text{registered\_at}_{\mathcal{I}_\mathcal{A}} = \{(\text{Tim, LU}), (\text{Bob, MU})\}$;
- $\text{student\_at}_{\mathcal{I}_\mathcal{A}} = \{(\text{Tom, Harvard})\}$.
In the table below, we consider Boolean queries $C(a)$ (in description logic notation!) and give the (certain) answer to $C(a)$ of the database instance $\mathcal{I}_A$, the ABox $\mathcal{A}$, and the knowledge base $\mathcal{K}_U = (\mathcal{T}_U, \mathcal{A})$.

<table>
<thead>
<tr>
<th>Boolean Query</th>
<th>$\mathcal{I}_A$</th>
<th>Abox $\mathcal{A}$</th>
<th>KB $\mathcal{K}_U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>University(CMU)</td>
<td>No</td>
<td>Don’t know</td>
<td>Yes</td>
</tr>
<tr>
<td>University(Harvard)</td>
<td>No</td>
<td>Don’t know</td>
<td>Yes</td>
</tr>
<tr>
<td>NonBritishUni(CMU)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Student(Tim)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Student(Tom)</td>
<td>No</td>
<td>Don’t know</td>
<td>Yes</td>
</tr>
<tr>
<td>$\exists$ student at. $\top$ (Tom)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$\exists$ student at. $\top$ (Tim)</td>
<td>No</td>
<td>Don’t know</td>
<td>Yes</td>
</tr>
<tr>
<td>(Student (\sqcap) ¬University)(Tim)</td>
<td>Yes</td>
<td>Don’t know</td>
<td>Yes</td>
</tr>
<tr>
<td>(Institution (\sqcap) ¬University)(FU Berlin)</td>
<td>Yes</td>
<td>Don’t know</td>
<td>Don’t know</td>
</tr>
</tbody>
</table>
Example

Let $S = (\mathcal{O}, \mathcal{B})$ be a knowledge base with simple ABox $\mathcal{B}$ given by

- Person(john), Person(nick), Person(toni)
- hasFather(john, nick), hasFather(nick, toni)

and TBox $\mathcal{O}$ defined as

$$\mathcal{O} = \{\text{Person} \sqsubseteq \exists \text{has\_Father\_Person}\}$$

For the FOPL query

$$F(x, y) = \text{hasFather}(x, y)$$

we obtain

$$\text{certanswer}(F, S) = \{(john, nick), (nick, toni)\}.$$
Example

• For the query

\[ F(x) = \exists y. \text{hasFather}(x, y) \]

we obtain

\[ \text{certanswer}(F(x), S) = \{\text{john, nick, toni}\} \]

• For the query

\[ F(x) = \exists y_1 \exists y_2 \exists y_3. (\text{hasFather}(x, y_1) \land \text{hasFather}(y_1, y_2) \land \text{hasFather}(y_2, y_3)) \]

we obtain

\[ \text{certanswer}(F(x), S) = \{\text{john, nick, toni}\} \]

• For the query

\[ F(x, y_3) = \exists y_1 \exists y_2. (\text{hasFather}(x, y_1) \land \text{hasFather}(y_1, y_2) \land \text{hasFather}(y_2, y_3)) \]

we obtain

\[ \text{certanswer}(F(x, y_3), S) = \emptyset \]
Complexity of querying \((\mathcal{T}, \mathcal{A})\)

Consider, for simplicity, Boolean queries. There are two different ways of measuring the complexity of querying:

- **Data complexity**: Measures the time/space needed to evaluate a fixed query \(F\) for a fixed TBox \(\mathcal{T}\) in \((\mathcal{T}, \mathcal{A})\) (i.e., check \(\mathcal{T}, \mathcal{A} \models F\)). The only input variable is the size of \(\mathcal{A}\).

- **Combined complexity**: Measure the time/space needed to evaluate a query in \((\mathcal{T}, \mathcal{A})\). The input variables are the size of the query, the size of \(\mathcal{T}\), and the size of \(\mathcal{A}\).

Data complexity is relevant if \(\mathcal{T}\) and the query are very small compared to \(\mathcal{A}\). This is the case in most applications.
Non-Tractability of Query answering in $\mathcal{ALC}$ in Data Complexity

A graph $G$ is a pair $(W, E)$ consisting of a set $W$ and a symmetric relation $E$ on $W$.

$G$ is 3-colorable if there exist subsets blue, red, and green of $W$ such that

- the sets blue, green, and red are mutually disjoint;
- blue $\cup$ red $\cup$ green $= W$;
- if $(a, b) \in E$, then $a$ and $b$ do not have the same color.

3-colorability of graphs is an NP-complete problem.
3-Colorability as a Query Answering Problem

Assume \( G = (W, E) \) is given. Construct the ABox \( \mathcal{A}_G \) by taking a role name \( r \) and setting

\[
\text{• } r(a, b) \in \mathcal{A} \text{ for all } a, b \in W \text{ with } (a, b) \in E.
\]

Construct the TBox \( \mathcal{ALC} \) TBox \( \mathcal{T}_C \) by taking concept names \( \text{Blue}, \text{Green}, \text{Red} \) and taking the inclusions:

\[
\text{• } \top \sqsubseteq \text{Blue} \sqcup \text{Green} \sqcup \text{Red}
\]

\[
\text{• } \text{Blue} \sqcap \exists r.\text{Blue} \sqsubseteq \text{Clash}
\]

\[
\text{• } \text{Red} \sqcap \exists r.\text{Red} \sqsubseteq \text{Clash}
\]

\[
\text{• } \text{Green} \sqcap \exists r.\text{Green} \sqsubseteq \text{Clash}
\]

Let \( F = \exists x \text{ Clash}(x) \). Then \( (\mathcal{T}_C, \mathcal{A}_G) \models F \) if, and only if, \( G \) is not 3-colorable.
Restricting the Description Logic and the Query Language

- FOPL is too expressive as a query language for knowledge bases. The combined complexity of querying even DL-Lite or $\mathcal{EL}$ knowledge bases with FOPL queries is undecidable.

- For $\mathcal{ALC}$ knowledge bases and basic Boolean queries of the form $\exists x A(x)$, ($A$ a concept name) query answering is still non-tractable. The best algorithms for query answering in this case are extensions of the $\mathcal{ALC}$ tableaux algorithms discussed above.

- We consider
  - knowledge bases in $\mathcal{EL}$, restricted Schema.org, and DL-Lite only;
  - queries in $\mathcal{EL}$ and conjunctive queries only.
Answering $EL$-Queries in $EL$ Knowledge Bases
**EL Concept Queries**

An **EL concept query** is a Boolean query of the form

\[ C(a) \]

where \( C \) is an **EL**-concept and \( a \) an individual name. We develop a method for answering **EL** concept queries in knowledge bases

\[ (T, A), \]

where \( T \) is an **EL**-TBox and \( A \) a simple ABox.

Note: Then we also have a method for computing

\[ \text{certanswer}(C(x), (T, A)) = \{ a \mid (T, A) \models C(a) \} \]
Fundamental Idea: reduce knowledge base querying to relational database querying

To answer the question whether

\[(\mathcal{T}, \mathcal{A}) \models C(a)\]

we construct from \((\mathcal{T}, \mathcal{A})\) a finite interpretation \(\mathcal{I}_{\mathcal{T}, \mathcal{A}}\) such that

\[(\mathcal{T}, \mathcal{A}) \models C(a) \iff \mathcal{I}_{\mathcal{T}, \mathcal{A}} \models C(a).\]

Thus, we reduce ontology based reasoning to database querying. After this construction database technology can be used to process queries.

Note: Such a reduction works only for a very limited number of ontology and query languages!
From $(\mathcal{T}, \mathcal{A})$ to $\mathcal{I}_{\mathcal{T}, \mathcal{A}}$

The algorithm constructing $\mathcal{I}_{\mathcal{T}, \mathcal{A}}$ is a rather simple extension of the algorithm deciding concept subsumption $A \sqsubseteq_B \mathcal{T}$ for $\mathcal{EL}$.

Firstly, we assume again that $\mathcal{T}$ is in normal form: it consists of inclusions of the form

- $A \sqsubseteq B$, where $A$ and $B$ are concept names;
- $A_1 \sqcap A_2 \sqsubseteq B$, where $A_1, A_2, B$ are concept names;
- $A \sqsubseteq \exists r.B$, where $A, B$ are concept names;
- $\exists r. A \sqsubseteq B$, where $A, B$ are concept names.
General Description

The domain $\Delta_{IT,A}$ of $IT,A$ consists of

- all individual names $a$ that occur in $A$;
- objects $d_A$, for every concept name $A$ in $T$. (In the description of the subsumption algorithm $d_A$ is denoted by $A$!)

It remains to compute

- $r_{IT,A}$, for all role names $r$;
- $A_{IT,A}$, for all concept names $A$.

This is done by computing functions $S$ and $R$ that are very similar to the functions introduced in the subsumption algorithm.
Algorithm Computing $\mathcal{I}_{\mathcal{T},\mathcal{A}}$

Given $\mathcal{T}$ in normal form and ABox $\mathcal{A}$, we compute functions $S$ and $R$:

- $S$ maps every $d \in \Delta^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ to a set $S(d)$ of concept names. We then set $d \in A^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ if $A \in S(d)$;

- $R$ maps every role name $r$ to a set $R(r)$ of pairs $(d_1, d_2)$ in $\Delta^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$. We then set $(d_1, d_2) \in r^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ if $(d_1, d_2) \in R(r)$.

We initialise $S$ and $R$ as follows:

- $S(a) = \{B \mid B(a) \in \mathcal{A}\}$;

- $S(d_A) = \{A\}$ (as in the subumption algorithm, where we had $d_A = A$!)

- $R(r) = \{(a, b) \mid r(a, b) \in \mathcal{A}\}$. 
Algorithm

Apply the following four rules to $S$ and $R$ exhaustively:

(simpleR) If $A \in S(d)$ and $A \sqsubseteq B \in \mathcal{T}$ and $B \not\in S(d)$, then

$$S(d) := S(d) \cup \{B\}.$$

(conjR) If $A_1, A_2 \in S(d)$ and $A_1 \cap A_2 \sqsubseteq B \in \mathcal{T}$ and $B \not\in S(d)$, then

$$S(d) := S(d) \cup \{B\}.$$

(rightR) If $A \in S(d)$ and $A \sqsubseteq \exists r . B \in \mathcal{T}$ and $(d, d_B) \not\in R(r)$, then

$$R(r) := R(r) \cup \{(d, d_B)\}.$$

(leftR) If $(d_1, d_2) \in R(r)$ and $B \in S(d_2)$ and $\exists r . B \sqsubseteq A \in \mathcal{T}$ and $A \not\in S(d_1)$, then

$$S(d_1) := S(d_1) \cup \{A\}.$$
Example

Let $\mathcal{T}$ be defined as:

- $\text{BasketballClub} \sqsubseteq \text{Club}$
- $\text{BasketballPlayer} \sqsubseteq \exists \text{plays_for}.\text{BasketballClub}$
- $\exists \text{plays_for}.\text{Club} \sqsubseteq \text{Player}$
- $\text{Player} \sqsubseteq \text{Human\_being}$

Let $\mathcal{A}$ be defined as:

- $\text{Basketballplayer}(\text{bob}), \text{Player}(\text{jim})$
- $\text{Basketballclub}(\text{tigers}), \text{Club}(\text{lions})$
- $\text{plays\_for}(\text{rob, tigers}), \text{plays\_for}(\text{bob, lions})$
Construction of $\mathcal{I}_T,A$

The initial assignment (with obvious abbreviations) is given by

$$S(d_{Basketclub}) = \{Basketclub\}$$

$$S(d_{Basketplayer}) = \{Basketplayer\}$$

$$S(d_{Club}) = \{Club\}$$

$$S(d_{Player}) = \{Player\}$$

$$S(d_{Human}) = \{Human\}$$

$$R(plays\_for) = \{(rob, tigers), (bob, lion)\}$$

$$S(bob) = \{Baskplayer\}$$

$$S(jim) = \{Player\}$$

$$S(tigers) = \{Baskclub\}$$

$$S(lions) = \{Club\}$$

$$S(rob) = \emptyset$$
Rule Applications

Now applications of (simpleR), (rightR), (leftR) are step-by-step as follows:

- Update $S$ using (simpleR):
  \[ S(d_{\text{BaskClub}}) = \{ \text{BaskClub, Club} \}. \]

- Update $R$ using (rightR):
  \[ R(\text{plays for}) = \{ (d_{\text{Baskplayer}}, d_{\text{BaskClub}}) \}. \]

- Update $S$ using (simpleR):
  \[ S(d_{\text{Player}}) = \{ \text{Player, Human} \}. \]

- Update $S$ using (leftR):
  \[ S(d_{\text{Baskplayer}}) = \{ \text{Baskplayer, Player} \}. \]

- Update $S$ using (simpleR):
  \[ S(d_{\text{Baskplayer}}) = \{ \text{Baskplayer, Player, Human} \}. \]
Rule applications continued

- Update $S$ using (simpleR):
  \[ S(\text{tigers}) = \{\text{BaskClub, Club}\}. \]

- Update $S$ using (simpleR):
  \[ S(\text{jim}) = \{\text{Player, Human}\}. \]

- Update $R$ using (rightR):
  \[ R(\text{plays for}) = \{(d_{\text{Baskplayer}}, d_{\text{BaskClub}}), (\text{bob}, d_{\text{BaskClub}})\}. \]

- Since $S(\text{bob})$ contains Baskplayer, we obtain using rules:
  \[ S(\text{bob}) = \{\text{Baskplayer, Player, Human}\}. \]

- Update $S$ using (leftR):
  \[ S(\text{rob}) = \{\text{Player}\}. \]

- Update $S$ using (leftR):
  \[ S(\text{rob}) = \{\text{Player, Human}\}. \]
The final assignment

\[ S(d_{\text{Baskclub}}) = \{\text{Baskclub, Club}\} \]
\[ S(d_{\text{Baskplayer}}) = \{\text{Baskplayer, Player, Human}\} \]
\[ S(d_{\text{Club}}) = \{\text{Club}\} \]
\[ S(d_{\text{Player}}) = \{\text{Player, Human}\} \]
\[ S(d_{\text{Human}}) = \{\text{Human}\} \]
\[ R(\text{plays for}) = \{(d_{\text{Baskplayer}}, d_{\text{BaskClub}}), (\text{rob, tigers}), (\text{bob, lion}), (\text{bob, } d_{\text{BaskClub}})\} \]
\[ S(\text{bob}) = \{\text{Baskplayer, Player, Human}\} \]
\[ S(\text{ jim}) = \{\text{Player}\} \]
\[ S(\text{tigers}) = \{\text{Baskclub}\} \]
\[ S(\text{lions}) = \{\text{Club}\} \]
\[ S(\text{rob}) = \{\text{Player, Human}\} \]
The interpretation $\mathcal{I}_{\mathcal{T}, \mathcal{A}}$

- $\Delta_{\mathcal{T}, \mathcal{A}} = \{d_{\text{Baskclub}}, d_{\text{Baskplayer}}, d_{\text{Club}}, d_{\text{Player}}, d_{\text{Human}}, \text{bob}, \text{jim}, \text{tigers}, \text{lions}, \text{rob}\}$;
- $\text{Baskclub}_{\mathcal{T}, \mathcal{A}} = \{d_{\text{Baskclub}}, \text{tigers}\}$;
- $\text{Club}_{\mathcal{T}, \mathcal{A}} = \{d_{\text{Club}}, d_{\text{Baskclub}}, \text{tigers}\}$;
- $\text{Baskplayer}_{\mathcal{T}, \mathcal{A}} = \{d_{\text{Baskplayer}}, \text{bob}\}$;
- $\text{Player}_{\mathcal{T}, \mathcal{A}} = \{d_{\text{Player}}, d_{\text{Baskplayer}}, \text{bob}, \text{jim}, \text{rob}\}$;
- $\text{Human}_{\mathcal{T}, \mathcal{A}} = \{d_{\text{Human}}, d_{\text{Player}}, d_{\text{Baskplayer}}, \text{bob}, \text{jim}, \text{rob}\}$;
- $\text{plays_for}_{\mathcal{T}, \mathcal{A}} = \{(d_{\text{Baskplayer}}, d_{\text{BaskClub}}), (\text{rob}, \text{tigers}), (\text{bob}, \text{lion}), (\text{bob}, d_{\text{BaskClub}})\}$.

Now

$$(\mathcal{T}, \mathcal{A}) \models C(a) \iff \mathcal{I}_{\mathcal{T}, \mathcal{A}} \models C(a)$$

for all $\mathcal{EL}$ concepts $C$ and $a$ in $\mathcal{A}$. For example,

$$\mathcal{I}_{\mathcal{T}, \mathcal{A}} \models \exists \text{plays_for.Baskclub}(\text{bob}), \quad \mathcal{I}_{\mathcal{T}, \mathcal{A}} \models \text{Human}(\text{rob})$$
Another Example

We consider the knowledge base $\mathcal{S} = (\mathcal{O}, \mathcal{B})$ given by the ABox $\mathcal{B}$ consisting of

$$\text{Person}(\text{john}), \text{Person}(\text{nick}), \text{Person}(\text{toni})$$

$$\text{hasFather}(\text{john}, \text{nick}), \text{hasFather}(\text{nick}, \text{toni})$$

and the TBox $\mathcal{O}$ given by

$$\mathcal{O} = \{ \text{Person} \sqsubseteq \exists \text{has_Father}.\text{Person} \}.$$

We construct $\mathcal{I}_S$. 
Constructing $\mathcal{I}_S$

The initial assignment is given by

$$S(d_{\text{Person}}) = \{\text{Person}\}$$

$$S(\text{john}) = \{\text{Person}\}$$

$$S(\text{nick}) = \{\text{Person}\}$$

$$S(\text{toni}) = \{\text{Person}\}$$

$$R(\text{hasFather}) = \{(\text{john}, \text{nick}), (\text{nick}, \text{toni})\}$$

Four applications of the rule (rightR) add

$$\{(\text{john}, d_{\text{Person}}), (\text{nick}, d_{\text{Person}}), (\text{toni}, d_{\text{Person}}), (d_{\text{Person}}, d_{\text{Person}})\}$$

to the original $R(\text{hasFather})$. After that, no rule is applicable.
The interpretation $\mathcal{I}_S$

We obtain the interpretation $\mathcal{I}_S$ defined as

$$\Delta^{\mathcal{I}_S} = \{d_{\text{Person}}, \text{john}, \text{nick}, \text{toni}\}$$

$$\text{Person}^{\mathcal{I}_S} = \{d_{\text{Person}}, \text{john}, \text{nick}, \text{toni}\}$$

$$\text{hasFather}^{\mathcal{I}_S} = \{(\text{john}, \text{nick}), (\text{nick}, \text{toni}), (\text{john}, d_{\text{Person}}),$$

$$\quad (\text{nick}, d_{\text{Person}}), (\text{toni}, d_{\text{Person}}), (d_{\text{Person}}, d_{\text{Person}})\}$$

We have

$$\mathcal{S} \models C(a) \iff \mathcal{I}_S \models C(a)$$

for all $\mathcal{EL}$ concepts $C$ and $a$ from $\mathcal{B}$. For example

$$\mathcal{I}_S \models \exists \text{hasFather}.\exists \text{hasFather}. \text{Person(toni)}$$
Answering Conjunctive Queries by Rewriting in DL-Lite
Conjunctive Queries

A FOPL query $F(x_1, \ldots, x_k)$ is a **conjunctive query** if it is constructed from atomic formulas $P(y_1, \ldots, y_n)$ using $\land$ and $\exists$ only.

In SQL, conjunctive queries correspond to

“Select-from-where queries”,

where the “where-conditions” use only conjunctions of “$=$-conditions”.

Ontology Languages
Examples

The queries

- \( F(x) = \text{Person}(x); \)
- \( F(x) = \exists y. \text{hasFather}(x,y); \)
- \( F(x) = \exists y_1 \exists y_2 \exists y_3. (\text{hasFather}(x,y_1) \land \text{hasFather}(y_1,y_2) \land \text{hasFather}(y_2,y_3)); \)
- \( F(x, y_3) = \exists y_1 \exists y_2. (\text{hasFather}(x,y_1) \land \text{hasFather}(y_1,y_2) \land \text{hasFather}(y_2,y_3)). \)

are conjunctive queries.
Query Rewriting for DL-Lite

Given a DL-Lite TBox $\mathcal{T}$ and a conjunctive query $F(x_1, \ldots, x_n)$ one can compute a FOPL query $F_{\mathcal{T}}(x_1, \ldots, x_n)$ such that for every simple ABox $\mathcal{A}$, the database instance $\mathcal{I}_\mathcal{A}$ corresponding to $\mathcal{A}$, and any $a_1, \ldots, a_n$ in $\text{Ind}(\mathcal{A})$ the following holds:

$$(\mathcal{T}, \mathcal{A}) \models F(a_1, \ldots, a_n) \iff \mathcal{I}_\mathcal{A} \models F_{\mathcal{T}}(a_1, \ldots, a_n).$$

Checking $\mathcal{I}_\mathcal{A} \models F_{\mathcal{T}}(a_1, \ldots, a_n)$ is again a standard database evaluation problem.

We first illustrate the construction of $F_{\mathcal{T}}(x_1, \ldots, x_n)$ using an example.
Example: Rewriting

For the TBox

$$T = \{\text{Basketballplayer} \sqsubseteq \text{Player}, \text{Footballplayer} \sqsubseteq \text{Player}, \text{Handballplayer} \sqsubseteq \text{Player}\}$$

and the query

$$F(x) = \text{Player}(x)$$

one can take

$$F_T(x) = \text{Basketballplayer}(x) \lor \text{Footballplayer}(x) \lor \text{Handballplayer}(x) \lor \text{Player}(x)$$
We give the rewriting algorithm for a small fragment $\text{DL-Lite}_\text{tiny}$ of $\text{DL-Lite}$ (and Schema.org) consisting of inclusions of the form

- $A \sqsubseteq B$, where $A$ and $B$ are concept names;
- domain restrictions $\exists r. \top \sqsubseteq A$, where $r$ is a role name and $A$ a concept name;
- range restrictions $\exists r^- . \top \sqsubseteq A$, where $r$ is a role name and $A$ a concept name.
Rewriting Algorithm for Fragment DL-Lite$_{\text{tiny}}$

The rewriting algorithm computes for any

- query of the form $F(x) = A(x)$ with $A$ a concept name and
- DL-Lite$_{\text{tiny}}$ TBox $\mathcal{T}$

a FOPL query $F_{\mathcal{T}}(x)$ such that for every simple ABox $\mathcal{A}$ and $a \in \text{Ind}(\mathcal{A})$:

$$ (\mathcal{T}, \mathcal{A}) \models A(a) \iff \mathcal{I}_\mathcal{A} \models F_{\mathcal{T}}(a) $$
The Algorithm

Assume $\mathcal{T}$ and $F(x) = A(x)$ are given. We compute sets $I(A)$, $I_R(A)$, and $I_{R^-}(A)$ which together provide ‘all possible reasons for $A(a)$’:

- Compute $I(A) = \{B \mid \mathcal{T} \models B \subseteq A\}$ as follows: Initialise $I(A) = \{A\}$. Now apply exhaustively the following rule: if $B' \in I(A)$ and $B \subseteq B' \in \mathcal{T}$ and $B \not\in I(A)$, then update

  $$I(A) := I(A) \cup \{B\}$$

- We obtain $I_R(A) = \{\exists r. \top \mid \mathcal{T} \models \exists r. \top \subseteq A\}$ as

  $$I_R(A) = \{\exists r. \top \mid \exists r. \top \subseteq B \in \mathcal{T}, B \in I(A)\}$$

- We obtain $I_{R^-}(A) = \{\exists r^- . \top \mid \mathcal{T} \models \exists r^- . \top \subseteq A\}$ as

  $$I_{R^-}(A) = \{\exists r^- . \top \mid \exists r^- . \top \subseteq B \in \mathcal{T}, B \in I(A)\}$$
Then set

\[ F_T(x) = \bigvee_{B \in I(A)} B(x) \lor \bigvee_{\exists r. \top \in I_R(A)} \exists y \forall r (x, y) \lor \bigvee_{\exists r. \top \in I_{R^-}(A)} \exists y \forall r (y, x) \]

Consider \( T \) defined as

\[ \exists \text{student at.} \top \sqsubseteq \text{Student}, \quad \exists \text{student at}. \top \sqsubseteq \text{University} \]

\[ \text{Student} \sqsubseteq \text{Person}, \quad \text{University} \sqsubseteq \text{Institution} \]

For \( F(x) = \text{Person}(x) \) we obtain

\[ F_T(x) = \text{Person}(x) \lor \text{Student}(x) \lor \exists y \text{student at}(x, y) \]