First-order Predicate Logic (FOPL) as an Ontology Language

Ontology Languages Based on First-order predicate Logic (FOPL)

The following standardized languages are based on FOPL:

- Common Logic
- CycL
- KIF

They are typically much more expressive than description logics and reasoning is typically undecidable. There are many upper level ontologies (domain independent ontologies that fix concepts across domains) formulated in FOPL. An example is DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering).

DOLCE

DOLCE defines general concepts such as 'participation', 'dependence', 'constitution' in an extension of FOPL.

For example, DOLCE contains axioms that fix the meaning of concepts related to 'parthood': Assume $\mathbf{P}(x, y)$ says that x is a part of y. Then

- $\mathbf{PP}(x, y) = \mathbf{P}(x, y) \wedge \neg \mathbf{P}(y, x)$ defines proper parthood;
- $\mathbf{O}(x, y) = \exists z (\mathbf{P}(z, x) \wedge \mathbf{P}(z, y))$ defines overlap;
- $\mathbf{AT}(x) = \neg \exists y \mathbf{PP}(y, x)$ defines atoms.

Syntax of FOPL: signature

A signature for FOPL consists of

- a set $\mathcal{V}$ of **variables** denoted by $x, x_0, x_1, \ldots, y, y_0, y_1, \ldots, z, z_0, z_1 \ldots$;
- a set $\mathcal{P}$ of **predicate symbols** each of which comes with a positive number as its arity. Predicates of arity one are called unary predicates and are often denoted by $A, A_0, A_1 \ldots, B, B_0, B_1, \ldots$ In description logic, they correspond to **concept names**, so we sometimes call them concept names. Predicates of arity two are called binary predicates and are often denoted by $r, r_0, r_1, \ldots, s, s_0, s_1, \ldots$. In description logic, they correspond to **roles**, so we sometimes call them roles.
- a "special" binary relation symbols "$=$".

Syntax of FOPL: formulas

Formulas of FOPL are defined by induction:

- If P is a predicate of arity k and $x_1, \ldots, x_k$ are terms, then $P(x_1, \ldots, x_k)$ is a formula;
- in particular, if x_1, x_2 are terms, then $x_1 = x_2$ is a formula;
- If F is a formula, then $\neg F$ is a formula;
- If F and G are formulas, then $F \wedge G$ and $F \vee G$ are formulas;
- If F is a formula and x a variable, then $\forall x.F$ and $\exists x.F$ are formulas.

We abbreviate

- $F_1 \rightarrow F_2 = \neg F_1 \vee F_2$;
- $F_1 \leftrightarrow F_2 = (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$.

Translation of concept inclusions into FOPL: idea

- $\textbf{Father} \equiv \textbf{Person} \sqcap \textbf{Male} \sqcap \exists \textbf{hasChild}.\top$.
- $\textbf{Student} \equiv \textbf{Person} \sqcap \exists \textbf{is\_registered\_at}.\textbf{University}$.
- $\textbf{Father} \sqsubseteq \textbf{Person}$.
- $\textbf{Father} \sqsubseteq \exists \textbf{hasChild}.\top$.

are translated to

- $\forall x.(\textbf{Father}(x) \leftrightarrow (\textbf{Person}(x) \wedge \textbf{Male}(x) \wedge \exists y.\textbf{hasChild}(x, y)))$.
- $\forall x.(\textbf{Student}(x) \leftrightarrow (\textbf{Person}(x) \wedge \exists y.(\textbf{is\_registered\_at}(x, y) \wedge \textbf{University}(y))))$.
- $\forall x.(\textbf{Father}(x) \rightarrow \textbf{Person}(x))$.
- $\forall x.(\textbf{Father}(x) \rightarrow \exists y.\textbf{hasChild}(x, y))$.

Semantics of FOPL: interpretation

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

- a non-empty set $\Delta^{\mathcal{I}}$ (the domain);
- an interpretation function that maps
  - every k-ary predicate symbol P to a k-ary relation over $\Delta^{\mathcal{I}}$. In particular,
    - every unary predicate symbol A is mapped to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and
    - every binary predicate symbol r is a mapped to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

This is almost the same definition as for description logics. The only difference is that we have predicate symbols of arity larger than 2.

Example

Consider

- the concept names **University**, **BritishUniversity**, **Student**,
- the role name **student_at**.

An interpretation $\mathcal{I}_U$ is given by

- $\Delta^{\mathcal{I}_U} = \{$**LU**, **MU**, **CMU**, **Tim**, **Tom**, **Rob**, $a, b, c\}$.
- $\mathbf{University}^{\mathcal{I}_U} = \{\mathbf{LU}, \mathbf{MU}, \mathbf{CMU}, a, b\}$;
- $\mathbf{BritishUniversity}^{\mathcal{I}_U} = \{\mathbf{LU}, \mathbf{MU}\}$;
- $\mathbf{Student}^{\mathcal{I}_U} = \{\mathbf{Tim}, \mathbf{Tom}, c\}$;
- $\mathbf{student\_at}^{\mathcal{I}_U} = \{(\mathbf{Tim}, \mathbf{LU}), (\mathbf{Rob}, \mathbf{CMU})\}$.

Semantics of FOPL: Satisfaction

A **variable assignment** α for an interpretation $\mathcal{I}$ is a function that maps every variable $x \in \mathcal{V}$ to an element $\alpha(x) \in \Delta^{\mathcal{I}}$.

We define the **satisfaction relation** "$\mathcal{I}, \alpha \models F$" between a formula F and an interpretation $\mathcal{I}$ w.r.t. an assignment α inductively as follows:

- $\mathcal{I}, \alpha \models P(x_1, \ldots, x_k)$ if, and only if, $(\alpha(x_1), \ldots, \alpha(x_k)) \in P^{\mathcal{I}}$;
- $\mathcal{I}, \alpha \models x_1 = x_2$ if, and only if, $\alpha(x_1) = \alpha(x_2)$;
- $\mathcal{I}, \alpha \models \neg F$ if, and only if, not $\mathcal{I}, \alpha \models F$;
- $\mathcal{I}, \alpha \models F \wedge G$ if, and only if, $\mathcal{I}, \alpha \models F$ and $\mathcal{I}, \alpha \models G$;
- $\mathcal{I}, \alpha \models F \vee G$ if, and only if, $\mathcal{I}, \alpha \models F$ or $\mathcal{I}, \alpha \models G$;

Satisfaction continued

- $\mathcal{I}, \alpha \models \exists x.F$ if, and only if, there exists a $d \in \Delta^{\mathcal{I}}$ such that $\mathcal{I}, \alpha[x \mapsto d] \models F$
- $\mathcal{I}, \alpha \models \forall x.F$ if, and only if, for all $d \in \Delta^{\mathcal{I}}$ we have $\mathcal{I}, \alpha[x \mapsto d] \models F$.

where $\alpha[x \mapsto d]$ stands for the assignment that coincides with α except that x is mapped to d.

A variable x is called **free** in a formula F if x has an occurrence in F such that no occurrence of $\exists x$ or $\forall x$ is in front of it. To indicate that the free variables in a formula F are among $x_1, \ldots, x_k$, we sometimes write $F(x_1, \ldots, x_k)$ instead of F.

To test whether $\mathcal{I}, \alpha \models F(x_1, \ldots, x_k)$ it is enough to know the $\alpha(x_1), \ldots, \alpha(x_k)$. Thus, we write

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

if $\mathcal{I}, \alpha \models F(x_1, \ldots, x_k)$ holds for some assignment (equivalently, all assignments) α with $\alpha(x_1) = a_1, \ldots, \alpha(x_k) = a_k$.

Example $\mathcal{I}_U$

- Let $\alpha(x) = \mathbf{LU}$. Then $\mathcal{I}_U, \alpha \models \mathbf{University}(x)$. We sometimes denote this by $\mathcal{I}_U \models \mathbf{University}(\mathbf{LU})$.

- Let $\alpha(x) = \mathbf{Tim}$. Then $\mathcal{I}_U, \alpha \not\models \mathbf{University}(x)$. We sometimes denote this by $\mathcal{I}_U \not\models \mathbf{University}(\mathbf{Tim})$.

- Let $\alpha(x) = \mathbf{Tim}$ and $\alpha(y) = \mathbf{LU}$. Then $\mathcal{I}_U, \alpha \models \mathbf{student\_at}(x, y)$. We sometimes denote this by $\mathcal{I}_U \models \mathbf{student\_at}(\mathbf{Tim}, \mathbf{LU})$.

- Let α be arbitrary. Then $\mathcal{I}_U, \alpha \models \exists x.\mathbf{University}(x)$; since this does not depend on α, we denote this by $\mathcal{I}_U \models \exists x.\mathbf{University}(x)$.

- We have $\mathcal{I}_U \models \exists x.\neg\mathbf{University}(x)$.

- We have $\mathcal{I}_U \models \neg\forall x.(\mathbf{Student}(x) \rightarrow \exists y.\mathbf{student\_at}(x, y))$.

FOPL Ontologies and Reasoning

A closed FOPL formula (also called a sentence) is a FOPL formula without free variables.

An **FOPL ontology** $\mathcal{T}$ is a finite set of closed FOPL formulas.

For example, the following translation $\mathbf{MED}^\sharp$ of the $\mathcal{EL}$ TBoxes MED is an FOPL ontology:

$$\begin{aligned}
\forall x.(\textbf{Pericardium}(x) &\rightarrow (\textbf{Tissue}(x) \wedge \exists y.(\textbf{cont_in}(x,y) \wedge \textbf{Heart}(y)))) \\
\forall x.(\textbf{Pericarditis}(x) &\rightarrow (\textbf{Inflammation}(x) \wedge \exists y.(\textbf{has_loc}(x,y) \wedge \textbf{Pericardium}(y)))) \\
\forall x.(\textbf{Inflammation}(x) &\rightarrow (\textbf{Disease}(x) \wedge \exists y.(\textbf{acts_on}(x,y) \wedge \textbf{Tissue}(y)))) \\
\forall x.(F(x) &\rightarrow (\textbf{Heartdisease}(x) \wedge \textbf{NeedTreatment}(x)))
\end{aligned}$$

where

$$F(x) = \textbf{Disease}(x) \wedge \exists y.(\textbf{has\_loc}(x,y) \wedge \exists z.(\textbf{cont\_in}(y,z) \wedge \textbf{Heart}(z)))$$

FOPL Ontologies and Reasoning

For an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ is a model of $\boldsymbol{T}$ (or, equivalently, that $\mathcal{I}$ satisfies $\boldsymbol{T}$) and write

$$\mathcal{I} \models \boldsymbol{T},$$

if, and only if, $\mathcal{I} \models \boldsymbol{F}$ for all $\boldsymbol{F} \in \boldsymbol{T}$.

An FOPL sentence $\boldsymbol{F}$ follows from a FOPL ontology $\boldsymbol{T}$, in symbols

$$\boldsymbol{T} \models \boldsymbol{F},$$

if, and only if, for all interpretations $\mathcal{I}$ the following holds: If $\mathcal{I} \models \boldsymbol{T}$, then $\mathcal{I} \models \boldsymbol{F}$.

For example, "Pericarditis needs treatment" follows from **MED** if, and only if,

$$\textbf{MED}^{\sharp} \models \forall x(\textbf{Percarditis}(x) \rightarrow \textbf{NeedsTreatment}(x)).$$

As mentioned above already, for FOPL ontologies $\boldsymbol{T}$ and FOPL sentences $\boldsymbol{F}$ the problem "$\boldsymbol{T} \models \boldsymbol{F}$" is undecidable!

First-order Predicate Logic (FOPL) as a Query Language

Data

A **ground sentence** F is of the form $P(c_1, \ldots, c_n)$, where P is an n-ary predicate and $c_1, \ldots, c_n$ are individual symbols.

A **database instance** is a finite set $\mathcal{D}$ of ground sentences. The set $\mathbf{Ind}(\mathcal{D})$ denote the set of individual symbols in $\mathcal{D}$.

For example, the following set $\mathcal{D}_{\mathsf{uni}}$ is a database instance:

- $\mathbf{University}(\mathbf{LU}), \quad \mathbf{University}(\mathbf{MU}), \quad \mathbf{University}(\mathbf{CMU})$;
- $\mathbf{BritishUniversity}(\mathbf{LU})$
- $\mathbf{Student}(\mathbf{Tim}), \quad \mathbf{Student}(\mathbf{Tom})$;
- $\mathbf{student\_at}(\mathbf{Tim}, \mathbf{LU}), \quad \mathbf{student\_at}(\mathbf{Rob}, \mathbf{CMU})$.

Here we have

$$\mathbf{Ind}(\mathcal{D}_{\mathsf{uni}}) = \{\mathbf{LU}, \mathbf{MU}, \mathbf{CMU}, \mathbf{Tim}, \mathbf{Tom}, \mathbf{Rob}\}$$

Querying Database Instances

An n**-ary query** Q is a mapping that takes as input a database instance $\mathcal{D}$ and outputs a set $Q(\mathcal{D})$ of n-tuples from $\mathbf{Ind}(\mathcal{D})$.

If $n = 0$, then Q is called a **Boolean query** and outputs either 'Yes' or 'No' or 'Don't know':

$$Q(\mathcal{D}) \in \{ \text{Yes, No, Don't know} \}.$$

Consider the database instance $\mathcal{D}_{\mathsf{uni}}$ with the following queries:

- Boolean Query: Is every British university a university?
- Boolean Query: Does every university have a student?
- Unary query: Output all students that are not students at any university.
- Unary query: Output all universities that are not British universities.

Assumptions

The answers to the queries above depend on our assumptions about the way in which $\mathcal{D}_{\mathsf{uni}}$ represents the world. The main options are:

- Closed World Assumption: if something is not stated explicitly in $\mathcal{D}_{\mathsf{uni}}$ it is false. We assume complete knowledge about the domain: if some ground sentence is true, it is included in $\mathcal{D}_{\mathsf{uni}}$.
- Open World Assumption: if something is not stated in $\mathcal{D}_{\mathsf{uni}}$, then we do not know whether it is true or false. We assume incomplete knowledge about the domain: if some ground sentence is not in $\mathcal{D}_{\mathsf{uni}}$, it can still be true.

In relational databases we make the closed world assumption!

In Schema.org Markup we make the Open World Assumption

Consider the following Schema.org Markup:

```
<div vocab="http://schema.org/" typeof="Movie">
  <h1 property="name">Avatar</h1>
  <div property="director" typeof="Person">
  Director: <span property="name">James Cameron</span>
 (born <time property="birthDate" datetime="1954-08-16">August 16, 1954</time>)
  </div>
  <span property="genre">Science fiction</span>
</div>
```

Corresponds to the database instance:

$$\mathbf{Movie}(m), \quad \mathbf{name}(o, \text{Avatar}), \quad \mathbf{director}(m, p)$$

$$\mathbf{name}(p, \text{Cameron}), \quad \mathbf{birthDay}(p, 1954), \quad \mathbf{genre}(m, \text{ScienceFiction})$$

Clearly, we cannot be sure that by collecting markup of webpages we collect all movies!

FOPL Relational Database Querying

We formulate queries as FOPL formulas. The query defined by F is n-ary if F has n free variables. A query is Boolean if it is defined by a closed formula (has no free variable).

We realize the closed world assumption by regarding a database instance as an interpretation.

The interpretation $\mathcal{I}_{\mathcal{D}}$ defined by a database instance $\mathcal{D}$ is given by setting:

- $\Delta^{\mathcal{I}_{\mathcal{D}}} = \textbf{Ind}(\mathcal{D})$;
- For any n-ary predicate P and any $(a_1, \ldots, a_n)$ in $\textbf{Ind}(\mathcal{D})$:

$$(a_1, \ldots, a_n) \in P^{\mathcal{I}_{\mathcal{D}}} \quad \text{if and only if} \quad P(a_1, \ldots, a_n) \in \mathcal{D}$$

Example

The interpretation $\mathcal{J} = \mathcal{I}_{\mathcal{D}_{\textsf{uni}}}$ defined by $\mathcal{D}_{\textsf{uni}}$ is given as follows:

- $\Delta^{\mathcal{J}} = \{\textsf{LU}, \textsf{MU}, \textsf{CMU}, \textsf{Tim}, \textsf{Tom}, \textsf{Rob}\}$;
- $\textsf{University}^{\mathcal{J}} = \{\textsf{LU}, \textsf{MU}, \textsf{CMU}\}$;
- $\textsf{BritishUniversity}^{\mathcal{J}} = \{\textsf{LU}\}$;
- $\textsf{Student}^{\mathcal{J}} = \{\textsf{Tim}, \textsf{Tom}\}$;
- $\textsf{student\_at}^{\mathcal{J}} = \{(\textsf{Tim}, \textsf{LU}), (\textsf{Rob}, \textsf{CMU})\}$.

FOPL Relational Database Queries (Closed World Semantics)

A tuple $(a_1, \ldots, a_n)$ in **Ind**$(\mathcal{D})$ is an **answer** to an FOPL query $F(x_1, \ldots, x_n)$ in database instance $\mathcal{D}$ if

$$\mathcal{I}_{\mathcal{D}} \models F(a_1, \ldots, a_n)$$

We set

$$\textbf{answer}(F(x_1, \ldots, x_n), \mathcal{D}) = \{(a_1, \ldots, a_n) \mid \mathcal{I}_{\mathcal{D}} \models F(a_1, \ldots, a_n)\}$$

The answer to a Boolean FOPL query F in $\mathcal{D}$ is 'Yes' if

$$\mathcal{I}_{\mathcal{D}} \models F$$

The answer to a Boolean FOPL query F in $\mathcal{D}$ is 'No' if

$$\mathcal{I}_{\mathcal{D}} \not\models F$$

Note that the answer 'Don't know' is never returned!

Examples

We query $\mathcal{D}_{\textbf{uni}}$ under closed world semantics:

- Output all universities. The corresponding query is

$$F(x) = \textbf{University}(x).$$

  The set of answers is $\textbf{answer}(F(x), \mathcal{D}_{\textbf{uni}}) = \{\textbf{LU}, \textbf{MU}, \textbf{CMU}\}$

- Is every British university a university? The corresponding query is

$$F = \forall x(\textbf{BritishUniversity}(x) \to \textbf{University}(x))$$

  The answer is 'Yes'.

Examples

- Does every university have a student? The corresponding query is

$$F = \forall x(\textbf{University}(x) \rightarrow \exists y \textbf{student\_at}(y, x))$$

  The answer is 'No'.

- Output all students that are not students at any university. The corresponding query is

$$F(x) = \textbf{Student}(x) \wedge \neg \exists y \textbf{student\_at}(x, y)$$

  The set of answers is $\textbf{answer}(F(x), \mathcal{D}_{\text{uni}}) = \{\textbf{Tom}\}$

- Output all universities that are not British universities. The corresponding query is

$$F(x) = \textbf{University}(x) \wedge \neg \textbf{BritishUniversity}(x)$$

  The set of answers is $\textbf{answer}(F(x), \mathcal{D}_{\text{uni}}) = \{\textbf{MU}, \textbf{CMU}\}$.

Complexity of querying relational databases

Consider, for simplicity, Boolean queries. There are two different ways of measuring the complexity of querying relational databases (RDBs):

- **Data complexity**: This measures the time/space needed (in the worst case) to evaluate a fixed query $\boldsymbol{F}$ in an instance $\mathcal{I}$ of a RDB (i.e., to decide whether $\mathcal{I} \models \boldsymbol{F}$). Thus, when measuring data complexity, the input variable is the size of the instance and the query is assumed to be fixed.

- **Combined Complexity**: This measures the time/space needed (in the worst case) to evaluate a query $\boldsymbol{F}$ in an instance $\mathcal{I}$ of a RDB. Thus, when measuring combined complexity, both the size of the instance and the query are input variables.

Data complexity is regarded as being more relevant as queries are typically rather small, but database instances are very large compared to the query.

FOPL: Data complexity is in LogSpace. So it is in polytime and, therefore, tractable. Combined complexity is PSpace-complete, and, therefore, not tractable.

FOPL Query Answering (Open World Semantics)

Let $\mathcal{D}$ be a database instance. An interpretation $\mathcal{I}$ is a model of $\mathcal{D}$ if

- $\mathbf{Ind}(\mathcal{D}) \subseteq \Delta^{\mathcal{I}}$;
- If $P(a_1, \ldots, a_n) \in \mathcal{D}$, then $(a_1, \ldots, a_n) \in P^{\mathcal{I}}$.

The set of models of $\mathcal{D}$ is denoted by $\mathbf{Mod}(\mathcal{D})$.

Let $F(x_1, \ldots, x_n)$ be an FOPL query. Then $(a_1, \ldots, a_n)$ in $\mathbf{Ind}(\mathcal{D})$ is a **certain answer** to $F(x_1, \ldots, x_n)$ in $\mathcal{D}$, in symbols

$$\mathcal{D} \models F(a_1, \ldots, a_n),$$

if $\mathcal{I} \models F(a_1, \ldots, a_n)$ for all $\mathcal{I} \in \mathbf{Mod}(\mathcal{D})$.

The set of certain answers to $F(x_1, \ldots, x_n)$ in $\mathcal{D}$ is

$$\mathbf{certanswer}(F(x_1, \ldots, x_n), \mathcal{D}) = \{(a_1, \ldots, a_n) \mid \mathcal{D} \models F(a_1, \ldots, a_n)\}$$

FOPL Query Answering (Open World Semantics)

- 'Yes' is the certain answer to a Boolean query $\boldsymbol{F}$ if $\mathcal{I} \models \boldsymbol{F}$ for all $\mathcal{I} \in \mathbf{Mod}(\mathcal{D})$.
- 'No' is the certain answer to a Boolean query $\boldsymbol{F}$ if $\mathcal{I} \not\models \boldsymbol{F}$ for all $\mathcal{I} \in \mathbf{Mod}(\mathcal{D})$
- If neither 'Yes' nor 'No' is a certain answer, then we say that the certain answer is 'Don't know'.

Note that under closed world semantics the answer to a Boolean query is always either 'Yes' or 'No'.

Example

We consider again the database instance $\mathcal{D}_{\text{uni}}$:

- $\textbf{University}(\textbf{LU}), \quad \textbf{University}(\textbf{MU}), \quad \textbf{University}(\textbf{CMU});$
- $\textbf{BritishUniversity}(\textbf{LU})$
- $\textbf{Student}(\textbf{Tim}), \quad \textbf{Student}(\textbf{Tom});$
- $\textbf{student\_at}(\textbf{Tim}, \textbf{LU}), \quad \textbf{student\_at}(\textbf{Rob}, \textbf{CMU}).$

Example

Output all universities. The corresponding query is

$$F(x) = \textbf{University}(x).$$

The set of certain answers is

$$\textbf{certanswer}(F(x), \mathcal{D}_{\textsf{uni}}) = \{\textbf{LU}, \textbf{MU}, \textbf{CMU}\}$$

This coincides with $\textbf{answer}(F(x), \mathcal{D}_{\textsf{uni}})$. To see this, we have to show:

- for all $\mathcal{I} \in \textbf{Mod}(\mathcal{D}_{\textsf{uni}})$: $\mathcal{I} \models F(\textbf{LU})$, $\mathcal{I} \models F(\textbf{MU})$, and $\mathcal{I} \models F(\textbf{CMU})$.
- there exists $\mathcal{I} \in \textbf{Mod}(\mathcal{D}_{\textsf{uni}})$ with $\mathcal{I} \not\models F(\textbf{Tim})$;
- there exists $\mathcal{I} \in \textbf{Mod}(\mathcal{D}_{\textsf{uni}})$ with $\mathcal{I} \not\models F(\textbf{Tom})$.
- there exists $\mathcal{I} \in \textbf{Mod}(\mathcal{D}_{\textsf{uni}})$ with $\mathcal{I} \not\models F(\textbf{Rob})$.

For Point 2 to 4 we just take $\mathcal{I}_{\mathcal{D}_{\textsf{uni}}}$!

Example

Is every British university a university? The corresponding query is

$$F = \forall x(\textbf{BritishUniversity}(x) \rightarrow \textbf{University}(x))$$

The certain answer to F in $\mathcal{D}_{\textbf{uni}}$ is 'Don't know'. To see this we have to construct two interpretations $\mathcal{I} \in \textbf{Mod}(\mathcal{D}_{\textbf{uni}})$ and $\mathcal{J} \in \textbf{Mod}(\mathcal{D}_{\textbf{uni}})$ such that

- $\mathcal{I} \models F$;
- $\mathcal{J} \not\models F$.

For Point 1 we can take $\mathcal{I}_{\mathcal{D}_{\textbf{uni}}}$. For Point 2, we expand $\mathcal{I}_{\mathcal{D}_{\textbf{uni}}}$ by adding a new individual a to its domain $\Delta^{\mathcal{J}}$ and let $a \in \textbf{BritishUniversity}^{\mathcal{J}}$ and $a \notin \textbf{University}^{\mathcal{J}}$.

Example

Does every university have a student? The corresponding query is

$$F = \forall x(\mathbf{University}(x) \rightarrow \exists y \mathbf{student\_at}(y, x))$$

The certain answer to F in $\mathcal{D}_{\mathsf{uni}}$ is 'Don't know'. To see this we have to construct two interpretations $\mathcal{I} \in \mathbf{Mod}(\mathcal{D}_{\mathsf{uni}})$ and $\mathcal{J} \in \mathbf{Mod}(\mathcal{D}_{\mathsf{uni}})$ such that

- $\mathcal{I} \models F$;
- $\mathcal{J} \not\models F$.

For Point 2 we can take $\mathcal{J} = \mathcal{I}_{\mathcal{D}_{\mathsf{uni}}}$ (since **MU** does not have a student in $\mathcal{I}_{\mathcal{D}_{\mathsf{uni}}}$). For Point 1, we expand $\mathcal{I}_{\mathcal{D}_{\mathsf{uni}}}$ to an interpretation $\mathcal{I}$ by adding a new individual a to its domain $\Delta^{\mathcal{I}}$ and let $(a, \mathbf{MU}) \in \mathbf{student\_at}^{\mathcal{I}}$.

Example

Output all students that are not students at any university. The corresponding query is

$$F(x) = \textbf{Student}(x) \land \neg \exists y \textbf{student\_at}(x, y)$$

The set of certain answers is $\textbf{certanswer}(F(x), \mathcal{D}_{\textbf{uni}}) = \emptyset$.

To see this, observe that $\mathcal{I}_{\mathcal{D}_{\textbf{uni}}}$ shows already that only **Tom** could be a certain answer. But **Tom** is not a certain answer since the interpretation $\mathcal{I}$ that expands $\mathcal{I}_{\mathcal{D}_{\textbf{uni}}}$ by adding $(\textbf{Tom}, \textbf{MU})$ to $\textbf{student\_at}^{\mathcal{I}}$ shows that there is a model of $\mathcal{D}_{\textbf{uni}}$ in which **Tom** is a student at some university.

Example

Output all universities that are not British universities. The corresponding query is

$$F(x) = \textbf{University}(x) \land \neg\textbf{BritishUniversity}(x)$$

The set of answers is $\textbf{certanswer}(F(x), \mathcal{D}_{\textbf{uni}}) = \emptyset$.

To see this, expand $\mathcal{I}_{\mathcal{D}_{\textbf{uni}}}$ by adding **MU** and **CMU** to $\textbf{BritishUniversity}^{\mathcal{I}}$. Then $\mathcal{I}$ is a model of $\mathcal{D}_{\textbf{uni}}$ in which there is no university that is not a British university.

The Relationship between Relational Databases and Ontologies

- FOPL database querying (closed world assumption) means checking

$$\mathcal{I} \models F$$

- FOPL and Description Logic ontology querying means checking

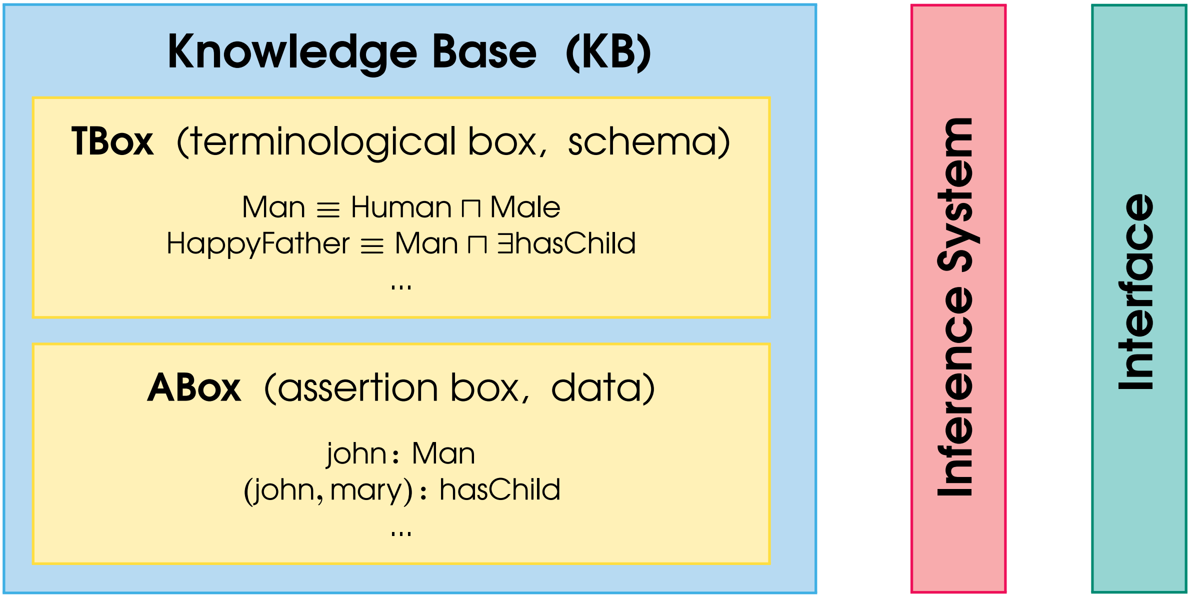
$$T \models F$$

- Typically, an FOPL or Description Logic ontology T has many different interpretations (many different $\mathcal{I}$ such that $\mathcal{I} \models T$). Data instances $\mathcal{D}$ have only one interpretation (under closed world assumption).

- FOPL and Description Logic ontologies make the open world assumption. For example, often individuals are not mentioned at all.

Using ABoxes to store Data

ABoxes (Assertion Boxes)



Assertion Box (ABox)

Let $\mathcal{L}$ be a description logic. A **$\mathcal{L}$-ABox** is a finite set $\mathcal{A}$ of assertions of the form

$$C(a), \quad r(a,b),$$

where C is an $\mathcal{L}$-concept, r a role name, and a, b are individual names.

- $C(a)$ says that a is an instance of C;
- $r(a,b)$ says that (a,b) is an instance of r.

ABoxes generalize database instances in which only ground sentences

$$A(a), \quad r(a,b)$$

with A a concept name and r a role name are allowed. We sometimes call ABoxes that are database instances **simple ABoxes**.

Semantics for ABoxes (Open World Assumption)

Let $\mathcal{A}$ be an ABox. By $\mathbf{Ind}(\mathcal{A})$ we denote the set of individual names in $\mathcal{A}$. An interpretation $\mathcal{I}$ is a model of $\mathcal{A}$, in symbols $\mathcal{I} \models \mathcal{A}$, if

- $\mathbf{Ind}(\mathcal{A}) \subseteq \Delta^{\mathcal{I}}$;
- If $C(a) \in \mathcal{A}$, then $a \in C^{\mathcal{I}}$;
- If $r(a, b) \in \mathcal{A}$, then $(a, b) \in r^{\mathcal{I}}$.

The set of models of $\mathcal{A}$ is denoted by $\mathbf{Mod}(\mathcal{A})$.

Let $F(x_1, \ldots, x_n)$ be an FOPL query. Then $(a_1, \ldots, a_n)$ in $\mathbf{Ind}(\mathcal{A})$ is a **certain answer** to $F(x_1, \ldots, x_n)$ in $\mathcal{A}$, in symbols

$$\mathcal{A} \models F(a_1, \ldots, a_n),$$

if $\mathcal{I} \models F(a_1, \ldots, a_n)$ for all $\mathcal{I} \in \mathbf{Mod}(\mathcal{A})$.

The set of certain answers to $F(x_1, \ldots, x_n)$ in $\mathcal{A}$ is

$$\mathbf{certanswer}(F(x_1, \ldots, x_n), \mathcal{A}) = \{(a_1, \ldots, a_n) \mid \mathcal{A} \models F(a_1, \ldots, a_n)\}$$

FOPL Query Answering (Open World Semantics)

- 'Yes' is the certain answer to a Boolean query $\boldsymbol{F}$ if $\mathcal{I} \models \boldsymbol{F}$ for all $\mathcal{I} \in \mathbf{Mod}(\mathcal{A})$.
- 'No' is the certain answer to a Boolean query $\boldsymbol{F}$ if $\mathcal{I} \not\models \boldsymbol{F}$ for all $\mathcal{I} \in \mathbf{Mod}(\mathcal{A})$
- If neither 'Yes' nor 'No' is a certain answer, then we say that the certain answer is 'Don't know'.

What is the answer to this query?

Consider the ABox $\mathcal{A}$:

1. $\mathsf{friend}(\mathsf{john}, \mathsf{susan})$
2. $\mathsf{friend}(\mathsf{john}, \mathsf{andrea})$
3. $\mathsf{loves}(\mathsf{susan}, \mathsf{andrea})$
4. $\mathsf{loves}(\mathsf{andrea}, \mathsf{bill})$
5. $\mathsf{Female}(\mathsf{susan})$
6. $\neg\mathsf{Female}(\mathsf{bill})$

> Does John have a female friend who is in love with a not female person?

The corresponding Boolean FOPL query is

$$F = \exists x.(\mathsf{friend}(\mathsf{john}, x) \wedge \mathsf{Female}(x) \wedge \exists y.(\mathsf{loves}(x, y) \wedge \neg\mathsf{Female}(y)))$$

or, in description logic notation:

$$\exists\mathsf{friend}.(\mathsf{Female} \sqcap \exists\mathsf{loves}.\neg\mathsf{Female})(\mathsf{john})$$

Answers: Example

Let

$$\mathcal{A} = \{ \text{Male(harry)}, \text{hasChild(peter, harry)} \}$$

The answer to the query "Are all children of Peter male?", in symbols

$$F = \forall x.(\text{hasChild}(\text{peter}, x) \to \text{Male}(x)),$$

given by $\mathcal{A}$ is "don't know".

In order to prevent this, we could add

- $\forall$hasChild.Male(peter)
- or $(\leq 1\,\text{hasChild}\,.\top)(\text{peter})$

to the ABox $\mathcal{A}$.

3-Colorability

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;
- $\mathbf{blue} \cup \mathbf{red} \cup \mathbf{green} = W$;
- 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 $\mathcal{A}$ by taking a role name r and concept names **Blue**, **Green**, and **Red** and setting

- $r(a, b) \in \mathcal{A}$ for all $a, b \in W$ with $(a, b) \in E$.
- $\textbf{Blue} \sqcup \textbf{Green} \sqcup \textbf{Red}(a) \in \mathcal{A}$ for all $a \in W$.
- $(\textbf{Blue} \rightarrow \forall r.(\textbf{Red} \sqcup \textbf{Green}))(a) \in \mathcal{A}$, for all $a \in W$;
- $(\textbf{Red} \rightarrow \forall r.(\textbf{Blue} \sqcup \textbf{Green}))(a) \in \mathcal{A}$, for all $a \in W$;
- $(\textbf{Green} \rightarrow \forall r.(\textbf{Red} \sqcup \textbf{Blue}))(a) \in \mathcal{A}$, for all $a \in W$.

Define query F by setting

$$F = \exists x((\textbf{Blue}(x) \wedge \textbf{Red}(x)) \vee (\textbf{Blue}(x) \wedge \textbf{Green}(x)) \vee (\textbf{Red}(x) \wedge \textbf{Green}(x))$$

Then G is not 3-colorable if, and only if, the certain answer to F in $\mathcal{A}$ is 'Yes'.

Thus, query answering is coNP-hard (the complement of NP) in data complexity!

Using the $\mathcal{ALC}$ Tableau to Answer Queries

Consider an $\mathcal{ALC}$ ABox $\mathcal{A}$ and a query of the form $C(x)$, where C is an $\mathcal{ALC}$ concept. Assume $a \in \mathbf{Ind}(\mathcal{A})$ is given. We want to know whether

$$a \in \mathbf{certanswer}(C(x), \mathcal{A}),$$

in other words, we want to know whether $a \in C^{\mathcal{I}}$ for all interpretations $\mathcal{I} \in \mathbf{Mod}(\mathcal{A})$.

We can reformulate this problem as follows: Let $\mathcal{A}' = \mathcal{A} \cup \{\neg C(a)\}$. Then $a \in \mathbf{certanswer}(C(x), \mathcal{A})$ if there does not exist any model of $\mathcal{A}'$.

Tableau Algorithm Deciding whether $\mathcal{A}$ has a Model

Consider $\mathcal{ALC}$ ABox $\mathcal{A}$. We may assume that each concept D in $\mathcal{A}$ is in negation normal form and obtain the constraint system $\mathcal{A}^*$ as the set of constraints

- $a : C$ for all $C(a) \in \mathcal{A}$;
- $(a, b) : r$ for all $r(a, b) \in \mathcal{A}$.

Then $\mathcal{A}$ has a model if, and only if, starting from $\mathcal{A}^*$ there is a sequence of completion rule applications that terminates with a set of constraints containing no clash.

Example

Consider again the ABox $\mathcal{A}$:

1. $\mathsf{friend}(\mathsf{john}, \mathsf{susan})$
2. $\mathsf{friend}(\mathsf{john}, \mathsf{andrea})$
3. $\mathsf{loves}(\mathsf{susan}, \mathsf{andrea})$
4. $\mathsf{loves}(\mathsf{andrea}, \mathsf{bill})$
5. $\mathsf{Female}(\mathsf{susan})$
6. $\neg\mathsf{Female}(\mathsf{bill})$

Does John have a female friend who is in love with a not female person?

Thus, we want to know whether 'Yes' is the certain answer to the query:

$$\exists \mathsf{friend}.(\mathsf{Female} \sqcap \exists \mathsf{loves}.\neg\mathsf{Female})(\mathsf{john})$$

Example

To this end we check whether

$$\mathcal{A} \cup \{\neg\exists \text{friend}.(\text{Female} \sqcap \exists \text{loves}.\neg\text{Female})(\text{john})\}$$

has a model. If not, then 'Yes' is indeed the certain answer to

$$\exists \text{friend}.(\text{Female} \sqcap \exists \text{loves}.\neg\text{Female})(\text{john})$$

Transformation into negation normal form gives:

$$\forall \text{friend}.(\neg\text{Female} \sqcup \forall \text{loves}.\text{Female})(\text{john})$$

Example

Thus, we apply the tableau to the constraint system

$$\mathcal{A}^* \cup \{\mathsf{john} : \forall \mathsf{friend}.(\neg \mathsf{Female} \sqcup \forall \mathsf{loves}.\mathsf{Female})\}$$

given by

1. $(\mathsf{john}, \mathsf{susan}) : \mathsf{friend}$
2. $(\mathsf{john}, \mathsf{andrea}) : \mathsf{friend}$
3. $(\mathsf{susan}, \mathsf{andrea}) : \mathsf{loves}$
4. $(\mathsf{andrea}, \mathsf{bill}) : \mathsf{loves}$
5. $\mathsf{susan} : \mathsf{Female}$
6. $\mathsf{bill} : \neg \mathsf{Female}$
7. $\mathsf{john} : \forall \mathsf{friend}.(\neg \mathsf{Female} \sqcup \forall \mathsf{loves}.\mathsf{Female})$

Example

Two applications of the rule $\rightarrow_\forall$ give the additional constraints:

$$\text{susan} : (\neg\text{Female} \sqcup \forall\text{loves}.\text{Female})$$

and

$$\text{andrea} : (\neg\text{Female} \sqcup \forall\text{loves}.\text{Female})$$

We now apply the rule $\rightarrow_\vee$ to the first constraint:

- Adding the constraint susan : ¬Female results in a clash since we have already susan : Female $\in \mathcal{A}^*$.
- Thus we add the constraint susan : ∀loves.Female to the constraint system.

Example

We now apply $\rightarrow_{\forall}$ to

$$\text{susan} : \forall\text{loves.Female}, \quad (\text{susan}, \text{andrea}) : \text{loves}$$

and add

$$\text{andrea} : \text{Female}$$

to the constraint system. We apply $\rightarrow_{\vee}$ to

$$\text{andrea} : (\neg\text{Female} \sqcup \forall\text{loves.Female})$$

- Adding andrea : $\neg$Female to the constraint systems results in a clash since andrea : Female is in the constraint system.
- Thus we add the constraint andrea : $\forall$loves.Female to the constraint system.

Example

Now we apply $\rightarrow_\forall$ to

$$\mathsf{andrea} : \forall \mathsf{loves}.\mathsf{Female}, \quad (\mathsf{andrea}, \mathsf{bill}) : \mathsf{loves}$$

and add

$$\mathsf{bill} : \mathsf{Female}$$

to the constraint system. But this results in a clash since $\mathsf{bill} : \neg\mathsf{Female}$ is already in the constraint system.

It follows that every sequence of completion rule application results in a clash.

Ontology Based Data Access

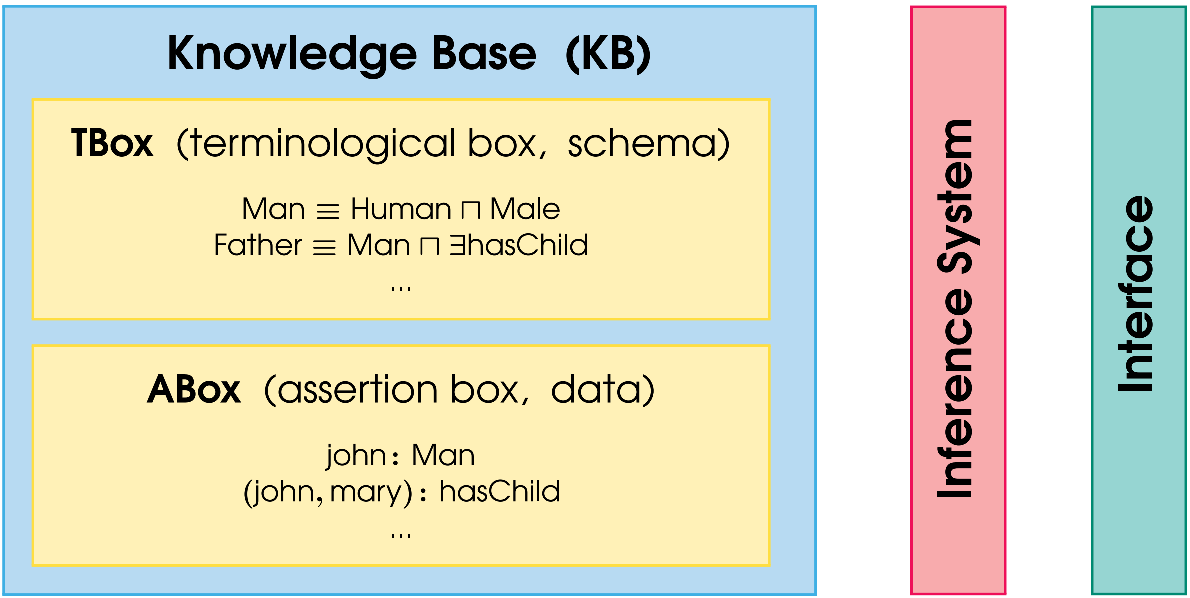
Vision: Ontologies at the Core of Information Systems

- 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)
$\text{Man} \equiv \text{Human} \sqcap \text{Male}$
$\text{Father} \equiv \text{Man} \sqcap \exists \text{hasChild}$
...
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 $\mathbf{Mod}(\mathcal{T}, \mathcal{A})$.

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 $\mathcal{I}$:

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

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

$$\textbf{certanswer}(F, \mathcal{K}) = \{(a_1, \ldots, a_k) \mid \mathcal{K} \models F(a_1, \ldots, a_k)\}$$

Boolean Queries

Let $\mathcal{K}$ be a knowledge base. For a query $\boldsymbol{F}$ without variables (Boolean query), we say that

- the certain answer given by $\mathcal{K}$ is "yes" if $\mathcal{I} \models \boldsymbol{F}$, for all interpretations $\mathcal{I}$ satisfying $\mathcal{K}$;
- the certain answer given by $\mathcal{K}$ is "no" if $\mathcal{I} \not\models \boldsymbol{F}$, for all interpretations $\mathcal{I}$ satisfying $\mathcal{K}$.
- Otherwise the certain answer is: "Don't know".

Example

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

- $\textbf{BritishUniversity} \sqsubseteq \textbf{University}$;
- $\textbf{University} \sqcap \textbf{Student} \sqsubseteq \bot$;
- $\top \sqsubseteq \forall \textbf{registered\_at}.\textbf{University}$;
- $\top \sqsubseteq \forall \textbf{student\_at}.\textbf{University}$;
- $\exists \textbf{student\_at}.\top \sqsubseteq \textbf{Student}$;
- $\textbf{Student} \sqsubseteq \exists \textbf{student\_at}.\top$;
- $\textbf{NonBritishUni} \equiv \textbf{University} \sqcap \neg \textbf{BritishUniversity}$.

Example (continued)

and the simple ABox (equivalently, database instance) $\mathcal{A}$:

- $\mathbf{NonBritishUni(CMU)}$
- $\mathbf{Institution(Harvard)}, \quad \mathbf{Institution(FUBerlin)}$
- $\mathbf{BritishUniversity(LU)}, \quad \mathbf{BritishUniversity(MU)}$
- $\mathbf{Student(Tim)}$
- $\mathbf{registered(Tim, LU)}, \quad \mathbf{registered(Bob, MU)}$
- $\mathbf{student\_at(Tom, Harvard)}$

Example (continued)

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

- $\Delta^{\mathcal{I}_\mathcal{A}} = \{\textbf{CMU}, \textbf{Harvard}, \textbf{FUBerlin}, \textbf{Tim}, \textbf{Tom}, \textbf{Bob}, \textbf{MU}, \textbf{LU}\}$;
- $\textbf{NonBritishUni}^{\mathcal{I}_\mathcal{A}} = \{\textbf{CMU}\}$;
- $\textbf{Institution}^{\mathcal{I}_\mathcal{A}} = \{\textbf{Harvard}, \textbf{FUBerlin}\}$;
- $\textbf{BritishUniversity}^{\mathcal{I}_\mathcal{A}} = \{\textbf{LU}, \textbf{MU}\}$;
- $\textbf{Student}^{\mathcal{I}_\mathcal{A}} = \{\textbf{Tim}\}$;
- $\textbf{registered\_at}^{\mathcal{I}_\mathcal{A}} = \{(\textbf{Tim}, \textbf{LU}), (\textbf{Bob}, \textbf{MU})\}$;
- $\textbf{student\_at}^{\mathcal{I}_\mathcal{A}} = \{(\textbf{Tom}, \textbf{Harvard})\}$.

(Certain) Answers

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 $\mathcal{I}_{\mathcal{A}}$, the ABox $\mathcal{A}$, and the knowledge base $\mathcal{K}_U = (\mathcal{T}_U, \mathcal{A})$.

Boolean Query	$\mathcal{I}_{\mathcal{A}}$	Abox $\mathcal{A}$	KB $\mathcal{K}_U$
University(**CMU**)	No	Don't know	Yes
University(**Harvard**)	No	Don't know	Yes
NonBritishUni(**CMU**)	Yes	Yes	Yes
Student(**Tim**)	Yes	Yes	Yes
Student(**Tom**)	No	Don't know	Yes
$\exists$**student_at**.$\top$(**Tom**)	Yes	Yes	Yes
$\exists$**student_at**.$\top$(**Tim**)	No	Don't know	Yes
(**Student** $\sqcap$ $\neg$**University**)(**Tim**)	Yes	Don't know	Yes
(**Institution** $\sqcap$ $\neg$**University**)(**FUBerlin**)	Yes	Don't know	Don't know

Example

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

$$\textbf{Person}(\textbf{john}), \quad \textbf{Person}(\textbf{nick}), \quad \textbf{Person}(\textbf{toni})$$

$$\textbf{hasFather}(\textbf{john}, \textbf{nick}), \quad \textbf{hasFather}(\textbf{nick}, \textbf{toni})$$

and TBox $\mathcal{O}$ defined as

$$\mathcal{O} = \{\textbf{Person} \sqsubseteq \exists \textbf{has\_Father}.\textbf{Person}\}$$

For the FOPL query

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

we obtain

$$\textbf{certanswer}(F, \mathcal{S}) = \{(\textbf{john}, \textbf{nick}), (\textbf{nick}, \textbf{toni})\}.$$

Example

- For the query
$$F(x) = \exists y.\textbf{hasFather}(x, y)$$
we obtain
$$\textbf{certanswer}(F(x), \mathcal{S}) = \{\textbf{john}, \textbf{nick}, \textbf{toni}\}$$

- For the query
$$F(x) = \exists y_1 \exists y_2 \exists y_3.(\textbf{hasFather}(x, y_1) \wedge \textbf{hasFather}(y_1, y_2) \wedge \textbf{hasFather}(y_2, y_3))$$
we obtain
$$\textbf{certanswer}(F(x), \mathcal{S}) = \{\textbf{john}, \textbf{nick}, \textbf{toni}\}$$

- For the query
$$F(x, y_3) = \exists y_1 \exists y_2.(\textbf{hasFather}(x, y_1) \wedge \textbf{hasFather}(y_1, y_2) \wedge \textbf{hasFather}(y_2, y_3))$$
we obtain
$$\textbf{certanswer}(F(x, y_3), \mathcal{S}) = \emptyset$$

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 $\boldsymbol{F}$ for a fixed TBox $\mathcal{T}$ in $(\mathcal{T}, \mathcal{A})$ (i.e., check $\mathcal{T}, \mathcal{A}) \models \boldsymbol{F}$). 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

- $r(a, b) \in \mathcal{A}$ for all $a, b \in W$ with $(a, b) \in E$.

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

- $\top \sqsubseteq \textbf{Blue} \sqcup \textbf{Green} \sqcup \textbf{Red}$
- $\textbf{Blue} \sqcap \exists r.\textbf{Blue} \sqsubseteq \textbf{Clash}$
- $\textbf{Red} \sqcap \exists r.\textbf{Red} \sqsubseteq \textbf{Clash}$
- $\textbf{Green} \sqcap \exists r.\textbf{Green} \sqsubseteq \textbf{Clash}$

Let $F = \exists x\ \textbf{Clash}(x)$. 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 $\mathcal{EL}$-Queries in $\mathcal{EL}$ Knowledge Bases

$\mathcal{EL}$ Concept Queries

An **$\mathcal{EL}$ concept query** is a Boolean query of the form

$$C(a)$$

where C is an $\mathcal{EL}$-concept and a an individual name. We develop a method for answering $\mathcal{EL}$ concept queries in knowledge bases

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

where $\mathcal{T}$ is a $\mathcal{EL}$-TBox and $\mathcal{A}$ a simple ABox.

Note: Then we also have a method for computing

$$\textbf{certanswer}(C(x), (\mathcal{T}, \mathcal{A})) = \{a \mid (\mathcal{T}, \mathcal{A}) \models C(a)\}$$

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

$$(\mathcal{T}, \mathcal{A}) \models C(a) \quad \Leftrightarrow \quad \mathcal{I}_{\mathcal{T},\mathcal{A}} \models C(a).$$

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 $A \sqsubseteq_{\mathcal{T}} B$ 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 $\Delta^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ of $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ consists of

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

It remains to compute

- $r^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$, for all role names r;
- $A^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$, 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 $\boldsymbol{S}$ and $\boldsymbol{R}$ exhaustively:

(simpleR) If $A \in S(d)$ and $A \sqsubseteq B \in \mathcal{T}$ and $B \notin S(d)$, then

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

(conjR) If $A_1, A_2 \in S(d)$ and $A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{T}$ and $B \notin S(d)$, then

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

(rightR) If $A \in S(d)$ and $A \sqsubseteq \exists r.B \in \mathcal{T}$ and $(d, d_B) \notin R(r)$, 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 $A \notin S(d_1)$, then

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

Example

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

$$\begin{array}{rcl} \textbf{BasketballClub} & \sqsubseteq & \textbf{Club} \\ \textbf{BasketballPlayer} & \sqsubseteq & \exists \textbf{plays\_for}.\textbf{BasketballClub} \\ \exists \textbf{plays\_for}.\textbf{Club} & \sqsubseteq & \textbf{Player} \\ \textbf{Player} & \sqsubseteq & \textbf{Human\_being} \end{array}$$

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

$$\textbf{Basketballplayer}(\textbf{bob}), \quad \textbf{Player}(\textbf{jim})$$

$$\textbf{Basketballclub}(\textbf{tigers}), \quad \textbf{Club}(\textbf{lions})$$

$$\textbf{plays\_for}(\textbf{rob}, \textbf{tigers}), \quad \textbf{plays\_for}(\textbf{bob}, \textbf{lions})$$

Construction of $\mathcal{I}_{\mathcal{T},\mathcal{A}}$

The initial assignment (with obvious abbreviations) is given by

$$
\begin{aligned}
S(d_{\textsf{Basketclub}}) &= \{\textsf{Basketclub}\} \\
S(d_{\textsf{Basketplayer}}) &= \{\textsf{Basketplayer}\} \\
S(d_{\textsf{Club}}) &= \{\textsf{Club}\} \\
S(d_{\textsf{Player}}) &= \{\textsf{Player}\} \\
S(d_{\textsf{Human}}) &= \{\textsf{Human}\} \\
R(\textsf{plays\_for}) &= \{(\textsf{rob}, \textsf{tigers}), (\textsf{bob}, \textsf{lion})\} \\
S(\textsf{bob}) &= \{\textsf{Baskplayer}\} \\
S(\textsf{jim}) &= \{\textsf{Player}\} \\
S(\textsf{tigers}) &= \{\textsf{Baskclub}\} \\
S(\textsf{lions}) &= \{\textsf{Club}\} \\
S(\textsf{rob}) &= \emptyset
\end{aligned}
$$

Rule Applications

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

- Update S using (simpleR):
$$S(d_{\textsf{BaskClub}}) = \{\textsf{BaskClub}, \textsf{Club}\}.$$
- Update R using (rightR):
$$R(\textsf{plays\_for}) = \{(d_{\textsf{Baskplayer}}, d_{\textsf{BaskClub}})\}.$$
- Update S using (simpleR):
$$S(d_{\textsf{Player}}) = \{\textsf{Player}, \textsf{Human}\}.$$
- Update S using (leftR):
$$S(d_{\textsf{Baskplayer}}) = \{\textsf{Baskplayer}, \textsf{Player}\}.$$
- Update S using (simpleR):
$$S(d_{\textsf{Baskplayer}}) = \{\textsf{Baskplayer}, \textsf{Player}, \textsf{Human}\}.$$

Rule applications continued

- Update S using (simpleR):

$$S(\textbf{tigers}) = \{\textbf{BaskClub}, \textbf{Club}\}.$$

- Update S using (simpleR):

$$S(\textbf{jim}) = \{\textbf{Player}, \textbf{Human}\}.$$

- Update R using (rightR):

$$R(\textbf{plays\_for}) = \{(d_{\textbf{Baskplayer}}, d_{\textbf{BaskClub}}), (\textbf{bob}, d_{\textbf{BaskClub}})\}.$$

- Since $S(\textbf{bob})$ contains **Baskplayer**, we obtain using rules:

$$S(\textbf{bob}) = \{\textbf{Baskplayer}, \textbf{Player}, \textbf{Human}\}.$$

- Update S using (leftR):

$$S(\textbf{rob}) = \{\textbf{Player}\}.$$

- Update S using (leftR):

$$S(\textbf{rob}) = \{\textbf{Player}, \textbf{Human}\}.$$

The final assignment

$$
\begin{aligned}
S(d_{\textsf{Baskclub}}) &= \{\textsf{Baskclub}, \textsf{Club}\} \\
S(d_{\textsf{Baskplayer}}) &= \{\textsf{Baskplayer}, \textsf{Player}, \textsf{Human}\} \\
S(d_{\textsf{Club}}) &= \{\textsf{Club}\} \\
S(d_{\textsf{Player}}) &= \{\textsf{Player}, \textsf{Human}\} \\
S(d_{\textsf{Human}}) &= \{\textsf{Human}\} \\
R(\textsf{plays\_for}) &= \{(d_{\textsf{Baskplayer}}, d_{\textsf{BaskClub}}), (\textsf{rob}, \textsf{tigers}), (\textsf{bob}, \textsf{lion}), (\textsf{bob}, d_{\textsf{BaskClub}})\} \\
S(\textsf{bob}) &= \{\textsf{Baskplayer}, \textsf{Player}, \textsf{Human}\} \\
S(\textsf{jim}) &= \{\textsf{Player}\} \\
S(\textsf{tigers}) &= \{\textsf{Baskclub}\} \\
S(\textsf{lions}) &= \{\textsf{Club}\} \\
S(\textsf{rob}) &= \{\textsf{Player}, \textsf{Human}\}
\end{aligned}
$$

The interpretation $\mathcal{I}_{\mathcal{T},\mathcal{A}}$

- $\Delta^{\mathcal{I}}_{\mathcal{T},\mathcal{A}} = \{d_{\textsf{Baskclub}}, d_{\textsf{Baskplayer}}, d_{\textsf{Club}}, d_{\textsf{Player}}, d_{\textsf{Human}}, \textbf{bob}, \textbf{jim}, \textbf{tigers}, \textbf{lions}, \textbf{rob}\}$;
- $\textbf{Baskclub}^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} = \{d_{\textsf{Baskclub}}, \textbf{tigers}\}$;
- $\textbf{Club}^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} = \{d_{\textsf{Club}}, d_{\textsf{Baskclub}}, \textbf{tigers}\}$;
- $\textbf{Baskplayer}^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} = \{d_{\textsf{Baskplayer}}, \textbf{bob}\}$;
- $\textbf{Player}^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} = \{d_{\textsf{Player}}, d_{\textsf{Baskplayer}}, \textbf{bob}, \textbf{jim}, \textbf{rob}\}$;
- $\textbf{Human}^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} = \{d_{\textsf{Human}}, d_{\textsf{Player}}, d_{\textsf{Baskplayer}}, \textbf{bob}, \textbf{jim}, \textbf{rob}\}$;
- $\textbf{plays\_for}^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} = \{(d_{\textsf{Baskplayer}}, d_{\textsf{BaskClub}}), (\textbf{rob}, \textbf{tigers}), (\textbf{bob}, \textbf{lion}), (\textbf{bob}, d_{\textsf{BaskClub}})\}$.

Now

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

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

$$\mathcal{I}_{\mathcal{T},\mathcal{A}} \models \exists \textbf{plays\_for}.\textbf{Baskclub}(\textbf{bob}), \quad \mathcal{I}_{\mathcal{T},\mathcal{A}} \models \textbf{Human}(\textbf{rob})$$

Another Example

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

$$\textbf{Person}(\textbf{john}), \quad \textbf{Person}(\textbf{nick}), \quad \textbf{Person}(\textbf{toni})$$

$$\textbf{hasFather}(\textbf{john}, \textbf{nick}), \quad \textbf{hasFather}(\textbf{nick}, \textbf{toni})$$

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

$$\mathcal{O} = \{\textbf{Person} \sqsubseteq \exists \textbf{has\_Father}.\textbf{Person}\}.$$

We construct $\mathcal{I}_{\mathcal{S}}$.

Constructing $\mathcal{I}_{\mathcal{S}}$

The initial assignment is given by

$$\begin{aligned} S(d_{\textsf{Person}}) &= \{\textsf{Person}\} \\ S(\textsf{john}) &= \{\textsf{Person}\} \\ S(\textsf{nick}) &= \{\textsf{Person}\} \\ S(\textsf{toni}) &= \{\textsf{Person}\} \\ R(\textsf{hasFather}) &= \{(\textsf{john}, \textsf{nick}), (\textsf{nick}, \textsf{toni})\} \end{aligned}$$

Four applications of the rule (rightR) add

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

to the original $R(\textsf{hasFather})$. After that, no rule is applicable.

The interpretation $\mathcal{I}_\mathcal{S}$

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

$$\begin{aligned}
\Delta^{\mathcal{I}_\mathcal{S}} &= \{d_{\mathsf{Person}}, \mathsf{john}, \mathsf{nick}, \mathsf{toni}\} \\
\mathsf{Person}^{\mathcal{I}_\mathcal{S}} &= \{d_{\mathsf{Person}}, \mathsf{john}, \mathsf{nick}, \mathsf{toni}\} \\
\mathsf{hasFather}^{\mathcal{I}_\mathcal{S}} &= \{(\mathsf{john}, \mathsf{nick}), (\mathsf{nick}, \mathsf{toni}), (\mathsf{john}, d_{\mathsf{Person}}), \\
&\quad (\mathsf{nick}, d_{\mathsf{Person}}), (\mathsf{toni}, d_{\mathsf{Person}}), (d_{\mathsf{Person}}, d_{\mathsf{Person}})\}
\end{aligned}$$

We have

$$\mathcal{S} \models C(a) \quad \Leftrightarrow \quad \mathcal{I}_\mathcal{S} \models C(a)$$

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

$$\mathcal{I}_\mathcal{S} \models \exists \mathsf{hasFather}.\exists \mathsf{hasFather}.\mathsf{Person}(\mathsf{toni})$$

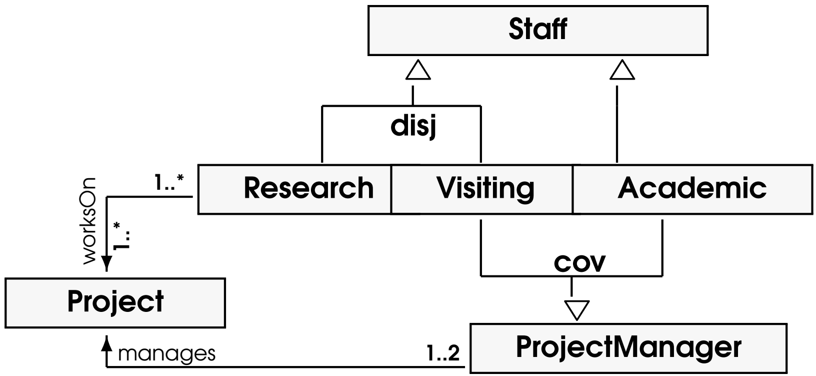
Answering Conjunctive Queries by Rewriting in DL-Lite

DL-Lite

DL-Lite is a family of description logics that has been designed to

- capture (at least) standard conceptual models such as ER and UML diagrams;
- provide a formal semantics for them;
- enable efficient access to data using a conceptual model represented as a DL-Lite TBox.

More details on what the last point means later. Here we introduce the terminological part of a member of the DL-Lite family which, for simplicity, we just call DL-Lite.



A UML diagram

How to represent a conceptual model such as this one in description logic?

DL-Lite (syntax)

- **Language for DL-Lite concepts (classes)**
  - concept names $A_0, A_1, \ldots$
  - role names $r_0, r_1, \ldots$
  - the concept $\top$ (often called "thing")
  - the concept $\bot$ (stands for the empty class)
  - the concept constructor $\sqcap$ (often called intersection, conjunction, or simply "and").
  - the concept constructor $\exists$ (often called existential restriction).
  - the concept constructor $\geq n$, where $n > 0$ is a natural number (called number restriction).
  - the role constructor $\cdot^-$ (called inverse role constructor).

DL-Lite

A **role** is either a role name or the inverse r^- over a role name r.

DL-Lite concepts are defined as follows:

- All concept names, $\top$ and $\bot$ are DL-Lite concepts;
- $\exists r.\top$ is a DL-Lite concept, for every role r;
- $(\geq n\ r.\top)$ is a DL-Lite concept, for every role r.
- If B_1 and B_2 are DL-Lite concepts, then $B_1 \sqcap B_2$ is a DL-Lite concept.

A **DL-Lite concept-inclusion** is of the form

$$B_1 \sqsubseteq B_2$$

where B_1 and B_2 are DL-Lite concepts.

A **DL-Lite TBox** is a finite set of DL-Lite concept inclusions.

Examples, discussion, and comparison with $\mathcal{EL}$

- $\mathbf{Person} \sqcap (\geq \mathbf{5}\ \mathbf{hasChild}.\top)$ (a person with at least 5 children);
- $\mathbf{Person} \sqcap (\geq \mathbf{7}\ \mathbf{hasChild}^{-}.\top)$ (a person with at least 7 parents (?));
- $\mathbf{Chair} \sqcap \mathbf{Table} \sqsubseteq \bot$ (the classes of chairs and tables are disjoint);
- The $\mathcal{EL}$-concept $\mathbf{Person} \sqcap \exists \mathbf{hasChild}.\mathbf{Person}$ cannot be expressed in DL-Lite;
- The $\mathcal{EL}$-concept $\mathbf{Person} \sqcap \exists \mathbf{hasChild}.\exists \mathbf{hasChild}.\top$ cannot be expressed in DL-Lite;
- $\exists \boldsymbol{r}.\top$ is equivalent to $(\geq \mathbf{1}\ \boldsymbol{r}.\top)$.

Question: Why don't we just put DL-Lite and $\mathcal{EL}$ together? Isn't this much better than introducing two languages...?

Domain Restriction in DL-Lite

The UML class diagram above says that

"everybody who manages something is a manager"

or, equivalently, "only managers manage something".

In other words, it states that every object in the domain of the relation "manages" is a Manager. This can be expressed in DL-Lite as

$$\exists \mathbf{manages}.\top \sqsubseteq \mathbf{Manager}$$

Inclusions of the form

$$\exists r.\top \sqsubseteq C$$

are called **domain restrictions**.

Range Restrictions in DL-Lite

The UML class diagram above says that

"everything that is managed by something is a project"

or, equivalently, "only projects are managed".

In other words, it states that every object in the range of the relation "manages" is a Project. This can be expressed in DL-Lite as

$$\exists \mathbf{manages}^{-}.\top \sqsubseteq \mathbf{Project}$$

Inclusions of the form

$$\exists r^{-}.\top \sqsubseteq C$$

are called **range restrictions**.

Disjointness Statements

The UML class diagram above says that

"nobody is in both classes Research and Visiting"

or, equivalently, the classes Research and Visiting are disjoint.

This can be expressed in DL-Lite as

$$\mathbf{Research} \sqcap \mathbf{Visiting} \sqsubseteq \bot$$

Inclusions of the form

$$A \sqcap B \sqsubseteq \bot$$

are called **disjointness statements**.

The UML class diagram above in DL-Lite (almost)

$$
\begin{array}{rclrcl}
\exists\,\mathsf{manages}.\top & \sqsubseteq & \mathsf{ProjectManager}, & \exists\,\mathsf{worksOn}.\top & \sqsubseteq & \mathsf{Research}, \\
\exists\,\mathsf{manages}^{-}.\top & \sqsubseteq & \mathsf{Project}, & \exists\,\mathsf{worksOn}^{-}.\top & \sqsubseteq & \mathsf{Project}, \\
\mathsf{Project} & \sqsubseteq & \exists\,\mathsf{manages}^{-}.\top, & \mathsf{Research} & \sqsubseteq & \exists\,\mathsf{worksOn}.\top, \\
(\geq \mathbf{3}\ \mathsf{manages}^{-}.\top) & \sqsubseteq & \bot, & \mathsf{Project} & \sqsubseteq & \exists\,\mathsf{worksOn}^{-}.\top, \\
\mathsf{Research} & \sqsubseteq & \mathsf{Staff}, & \mathsf{Visiting} & \sqsubseteq & \mathsf{Staff}, \\
\mathsf{Research} \sqcap \mathsf{Visiting} & \sqsubseteq & \bot, & \mathsf{Academic} & \sqsubseteq & \mathsf{Staff}, \\
\mathsf{Visiting} & \sqsubseteq & \mathsf{ProjectManager}, & \mathsf{Academic} & \sqsubseteq & \mathsf{ProjectManager}.
\end{array}
$$

The only missing information is the covering constraint that

every Projectmanager is VistingStaff or AcademicStaff.

This cannot be expressed in DL-Lite.

DL-Lite (semantics)

Interpretations are defined as before:

- Recall that an **interpretation** is a structure $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ in which
  - $\Delta^{\mathcal{I}}$ is the **domain** (a non-empty set)
  - $\cdot^{\mathcal{I}}$ is an **interpretation function** that maps:
    - every concept name A to a subseteq $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ $(A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}})$
    - every role name r to a binary relation $r^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$ $(r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$
- The interpretation of an inverse role $(r^-)^{\mathcal{I}}$ is the inverse of $r^{\mathcal{I}}$:

$$(r^-)^{\mathcal{I}} = \{(d, e) \mid (e, d) \in r^{\mathcal{I}}\}$$

DL-Lite (semantics)

The interpretation $B^{\mathcal{I}}$ of a DL-Lite concept B in $\mathcal{I}$ is defined inductively as follows:

- $(\top)^{\mathcal{I}} = \Delta^{\mathcal{I}}$;
- $(\bot)^{\mathcal{I}} = \emptyset$;
- $(\geq n \ r.\top)^{\mathcal{I}}$ is the set of all $x \in \Delta^{\mathcal{I}}$ such that the number of y in $\Delta^{\mathcal{I}}$ with $(x, y) \in r^{\mathcal{I}}$ is at least n;
- $(\exists r.\top)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid$ there exists $y \in \Delta^{\mathcal{I}}$ such that $(x, y) \in r^{\mathcal{I}}\}$;
- $(B_1 \sqcap B_2)^{\mathcal{I}} = B_1^{\mathcal{I}} \cap B_2^{\mathcal{I}}$.

Semantics: exactly the same as for $\mathcal{EL}$

Let $\mathcal{I}$ be an interpretation, $\boldsymbol{B_1} \sqsubseteq \boldsymbol{B_2}$ a DL-Lite inclusion, and $\mathcal{T}$ a DL-Lite TBox.

- $\mathcal{I} \models \boldsymbol{B_1} \sqsubseteq \boldsymbol{B_2}$ if, and only if, $\boldsymbol{B_1^{\mathcal{I}}} \subseteq \boldsymbol{B_2^{\mathcal{I}}}$. In words:
  - $\mathcal{I}$ satisfies $\boldsymbol{B_1} \sqsubseteq \boldsymbol{B_2}$ or
  - $\boldsymbol{B_1} \sqsubseteq \boldsymbol{B_2}$ is true in $\mathcal{I}$ or
  - $\mathcal{I}$ is a model of $\boldsymbol{B_1} \sqsubseteq \boldsymbol{B_2}$.
- We set $\mathcal{I} \models \mathcal{T}$ if, and only if, $\mathcal{I} \models \boldsymbol{B_1} \sqsubseteq \boldsymbol{B_2}$ for all $\boldsymbol{B_1} \sqsubseteq \boldsymbol{B_2}$ in $\mathcal{T}$. In words:
  - $\mathcal{I}$ satisfies $\mathcal{T}$ or
  - $\mathcal{I}$ is a model of $\mathcal{T}$.

Reasoning Services for DL-Lite

- **Subsumption**. Let $\mathcal{T}$ be a TBox and $B_1 \sqsubseteq B_2$ a DL-Lite concept inclusion. We say that **$B_1 \sqsubseteq B_2$ follows from $\mathcal{T}$** if, and only if, every model of $\mathcal{T}$ is a model of $B_1 \sqsubseteq B_2$. Again we often write
  - $\mathcal{T} \models B_1 \sqsubseteq B_2$ or
  - $B_1 \sqsubseteq_{\mathcal{T}} B_2$.
- **TBox Satisfiablility**. A TBox $\mathcal{T}$ is satisfiable if, and only if, there exists a model of $\mathcal{T}$.

Theorem. For DL-Lite, there exist polytime algorithms deciding subsumption and TBox satisfiability.

Question: Why didn't we consider the TBox satisfiability problem for $\mathcal{EL}$-TBoxes?

Examples

Let $\mathcal{T} = \{A \sqsubseteq (\geq 2\ r.\top)\}$. Then

$$\mathcal{T} \not\models A \sqsubseteq (\geq 3\ r.\top).$$

To see this, construct an interpretation $\mathcal{I}$ such that

- $\mathcal{I} \models \mathcal{T}$;
- $\mathcal{I} \not\models A \sqsubseteq (\geq 3\ r.\top)$.

Namely, let $\mathcal{I}$ be defined by

- $\Delta^{\mathcal{I}} = \{a, b, c\}$;
- $A^{\mathcal{I}} = \{a\}$;
- $r^{\mathcal{I}} = \{(a, b), (a, c)\}$;

Then $A^{\mathcal{I}} = \{a\} \subseteq \{a\} = (\geq 2\ r.\top)^{\mathcal{I}}$ and so $\mathcal{I} \models \mathcal{T}$. But $A^{\mathcal{I}} = \{a\} \not\subseteq \emptyset = (\geq 3\ r.\top)^{\mathcal{I}}$ and so $\mathcal{I} \not\models A \sqsubseteq (\geq 3\ r.\top)$.

Example: Satisfiability

Let

$$\mathcal{T} = \{A \sqsubseteq \exists r.\top, \exists r^{-}.\top \sqsubseteq B, \top \sqsubseteq A, B \sqsubseteq \bot\}$$

Then $\mathcal{T}$ is not satisfiable.

To see this, assume for a proof by contradiction that $\mathcal{I} \models \mathcal{T}$.

Let $a \in \Delta^{\mathcal{I}}$.

- Then $a \in \top^{\mathcal{I}}$ and so $a \in A^{\mathcal{I}}$.
- Hence $a \in (\exists r.\top)^{\mathcal{I}}$.
- Hence there exists $b \in \Delta^{\mathcal{I}}$ with $(a, b) \in r^{\mathcal{I}}$.
- Hence $b \in (\exists r^{-}.\top)^{\mathcal{I}}$.
- Hence $b \in B^{\mathcal{I}}$.
- But this contradicts $B \sqsubseteq \bot \in \mathcal{T}$.

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 $\wedge$ and $\exists$ only.

In SQL, conjunctive queries correspond to

"Select-from-where queries",

where the "where-conditions" use only conjunctions of "$=$-conditions".

Examples

The queries

- $F(x) = \textbf{Person}(x)$;
- $F(x) = \exists y.\textbf{hasFather}(x, y)$;
- $F(x) = \exists y_1 \exists y_2 \exists y_3.(\textbf{hasFather}(x, y_1) \wedge \textbf{hasFather}(y_1, y_2); \wedge \textbf{hasFather}(y_2, y_3))$,
- $F(x, y_3) = \exists y_1 \exists y_2.(\textbf{hasFather}(x, y_1) \wedge \textbf{hasFather}(y_1, y_2) \wedge \textbf{hasFather}(y_2, y_3))$.

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 $\mathbf{Ind}(\mathcal{A})$ the following holds:

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

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

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

and the query

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

one can take

$$F_{\mathcal{T}}(x) = \textbf{Basketballplayer}(x) \vee \textbf{Footballplayer}(x) \vee \textbf{Handballplayer}(x) \vee \textbf{Player}(x)$$

Rewriting Algorithm for Fragment DL-Lite$_{\text{tiny}}$

We give the rewriting algorithm for a small fragment DL-Lite$_{\text{tiny}}$ of 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 $a \in \mathbf{Ind}(\mathcal{A})$:

$$(\mathcal{T}, \mathcal{A}) \models A(a) \quad \Leftrightarrow \quad \mathcal{I}_{\mathcal{A}} \models F_{\mathcal{T}}(a)$$

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

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

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

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

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

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

The Algorithm

Then set

$$F_{\mathcal{T}}(x) = \bigvee_{B \in I(A)} B(x) \vee \bigvee_{\exists r.\top \in I_R(A)} \exists y r(x,y) \vee \bigvee_{\exists r.\top \in I_{R^-}(A)} \exists y r(y,x)$$

Consider $\mathcal{T}$ defined as

$$\exists \textbf{student\_at}.\top \sqsubseteq \textbf{Student}, \quad \exists \textbf{student\_at}^{-}.\top \sqsubseteq \textbf{University}$$

$$\textbf{Student} \sqsubseteq \textbf{Person}, \quad \textbf{University} \sqsubseteq \textbf{Institution}$$

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

$$F_{\mathcal{T}}(x) = \textbf{Person}(x) \vee \textbf{Student}(x) \vee \exists y \textbf{student\_at}(x,y)$$