Logik für Informatiker
Logic for computer scientists
Ontologies: Description Logics

Till Mossakowski

WiSe 2009/10
Ontology languages

- **description logics** (efficiently decidable fragments of first-order logic)
  - used for domain ontologies
  - standardised in web ontology language OWL
- **first-order logics**
  - used for upper ontologies
  - standardised in Common Logic, CASL
Semantic networks

- used for representation of and reasoning about knowledge
- e.g. KL-ONE: reasoning about concepts, subclassing and their relations

```
```

```
```
Description Logics

- drawback of semantic networks:
  - often, meaning of arrows is not precisely defined
  - sometimes, full first-order logic is used $\Rightarrow$ undecidable

- Description Logics:
  - completely formal syntax and semantics,
  - decidable fragments of first-order logic
  - efficient reasoning tools available (Pellet, Fact++, Racer)
Concepts (in OWL: classes) (Mother, Father, etc.)

Subsumption $C \sqsubseteq D$ (read: “$C$ is subsumed by $D$”) means that each $C$ is a $D$

- Woman $\sqsubseteq$ Person
- Father $\sqsubseteq$ Male
- ...
To relate concepts, we need \textit{roles} (in OWL: \textit{properties}) like 'hasChild'.

- $\text{Parent} \sqsubseteq \exists \text{hasChild}. \top$ ($\top$: top concept, includes everything. In OWL: \textit{Thing})
- $\text{Parent} \sqsubseteq \exists \text{hasChild}. \text{Child}$
- $\text{Child} \sqsubseteq \exists \text{hasParent}. \top$ (Bad, because \textit{hasChild} is converse to \textit{hasParent} which is not expressed here)
- $\text{Child} \sqsubseteq \exists \text{hasChild}^\sim. \top$ (Better formalization)
- $\text{hasParent} \equiv \text{hasChild}^\sim$ (Alternative, not possible in every DL)
- $\text{Grandfather} \equiv (\exists \text{hasChild}. \exists \text{hasChild}. \top) \sqcap \text{Male}$
  ($C \equiv D$ is an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$)
- $\text{Grandfather} \equiv (\exists \text{hasChild}. \text{Parent}) \sqcap \text{Father}$
  (Alternative formalization)
A $DL$-signature $\Sigma = (C, R, I)$ consists of
- a set $C$ of concept names,
- a set $R$ of role names,
- a set $I$ of individual names,
Description Logics: Concepts

For a signature $\Sigma = (C, R, I)$ the set of $\mathcal{ALC}$-concepts over $\Sigma$ is defined by the following grammar:

$$C ::= A \text{ for } A \in C$$

\[
\begin{align*}
| & \top \\
| & \bot \\
| & \neg C \\
| & C \sqcap C \\
| & C \sqcup C \\
| & \exists R.C \text{ for } R \in R \\
| & \forall R.C \text{ for } R \in R
\end{align*}
\]

$\mathcal{ALC}$ stands for “attributive language with complement”

(Hets) Manchester syntax

- a concept name
- a concept
- not $C$
- $C$ and $C$
- $C$ or $C$
- $R$ some $C$
- $R$ only $C$

Till Mossakowski

Logic
The set of $\mathcal{ALC}$-Sentences over $\Sigma$ ($\text{Sen}(\Sigma)$) is defined as

- $C \sqsubseteq D$, where $C$ and $D$ are $\mathcal{ALC}$-concepts over $\Sigma$.
  - Class: $C$  SubclassOf: $D$

- $a : C$, where $a \in I$ and $C$ is a $\mathcal{ALC}$-concept over $\Sigma$.
  - Individual: $a$  Types: $C$

- $R(a_1, a_2)$, where $R \in R$ and $a_1, a_2 \in I$.
  - Individual: $a_1$  Facts: $R$ $a_2$
Description logics axioms are generally split up in two sets:

- **TBox**: subsumptions and definitions involving concepts and roles
  
  - e.g. $\text{Woman} \sqsubseteq \text{Person}$

- **ABox**: individuals and their membership in concepts and roles
  
  - e.g. $\text{john} : \text{Father}, \text{hasChild(john, harry)}$
Given $\Sigma = (C, R, I)$, a $\Sigma$-model is of form $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where

- $\Delta^\mathcal{I}$ is a non-empty set
- $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ for each $A \in C$
- $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ for each $R \in R$
- $a^\mathcal{I} \in \Delta^\mathcal{I}$ for each $a \in I$
We can extend $\mathcal{I}$ to all concepts as follows:

$\top^\mathcal{I} = \Delta^\mathcal{I}$

$\bot^\mathcal{I} = \emptyset$

$(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}$

$(C \cap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$

$(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$

$(\exists R.C)^\mathcal{I} = \{ x \in \Delta^\mathcal{I} | \exists y \in \Delta^\mathcal{I}. (x, y) \in R^\mathcal{I}, y \in C^\mathcal{I} \}$

$(\forall R.C)^\mathcal{I} = \{ x \in \Delta^\mathcal{I} | \forall y \in \Delta^\mathcal{I}. (x, y) \in R^\mathcal{I} \Rightarrow y \in C^\mathcal{I} \}$
Description Logic: Satisfaction of sentences in a model

\[ \mathcal{I} \models C \subseteq D \quad \text{iff} \quad C^\mathcal{I} \subseteq D^\mathcal{I}. \]

\[ \mathcal{I} \models a : C \quad \text{iff} \quad a^\mathcal{I} \in C^\mathcal{I}. \]

\[ \mathcal{I} \models R(a_1, a_2) \quad \text{iff} \quad (a_1^\mathcal{I}, a_2^\mathcal{I}) \in R^\mathcal{I}. \]
For $\Gamma \subseteq \text{Sen}(\Sigma)$, $\phi \in \text{Sen}(\Sigma)$, $\phi$ is a logical consequence of $\Gamma$ (written: $\Gamma \models_{\Sigma} \phi$), if for each $\Sigma$-model $\mathcal{I}$

$$\mathcal{I} \models \Gamma \text{ implies } \mathcal{I} \models \phi.$$ 

If $\Gamma$ contains only subsumptions, $\Gamma$ is written as $\top$ (TBox).
If $\Gamma$ contains only sentences $a : C$ and $R(a_1, a_2)$, $\Gamma$ is written as $\mathcal{A}$ (ABox).
<table>
<thead>
<tr>
<th>Class</th>
<th>Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>VegetarianPizza</td>
<td>⊑ Pizza</td>
</tr>
<tr>
<td>MagheritaPizza</td>
<td>⊑ Pizza</td>
</tr>
<tr>
<td>TomatoTopping</td>
<td>⊑ VegetableTopping</td>
</tr>
<tr>
<td>MozzarellaTopping</td>
<td>⊑ CheeseTopping</td>
</tr>
<tr>
<td>VegetarianPizza</td>
<td>≡ ∀ hasTopping (VegetableTopping ∪ CheeseTopping)</td>
</tr>
<tr>
<td>MagheritaPizza</td>
<td>⊑ ∃ hasTopping MozarellaTopping ⊔ ∃ hasTopping TomatoTopping ⊔ ∀ hasTopping (MozarellaTopping ∪ TomatoTopping)</td>
</tr>
</tbody>
</table>

Logical consequence: MagheritaPizza ⊑ VegetarianPizza
Usually, satisfiability of concepts is tested. A concept \( C \) is *satisfiable* in a TBox iff there is a model of the TBox that leads to a non-empty interpretation of \( C \).

Satisfiability and subsumption are inter-reducible:

\[
\mathcal{T} \models C \sqsubseteq D \quad \text{iff} \quad \mathcal{T} \models \text{unsat}(C \sqcap \neg D)
\]

\[
\mathcal{T} \models \text{unsat}(C) \quad \text{iff} \quad \mathcal{T} \models C \sqsubseteq \bot
\]

Complexity of TBox reasoning for \( ALC \):

- general TBoxes: EXPTIME complete
- empty or acyclic TBoxes: PSPACE complete\(^1\).

Acyclic TBoxes contain only definitions \( A \equiv C \), such that concept dependency is acyclic (\( A \) depends on all concepts occurring in \( C \)).

---

\(^1\)We know that \( P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \) and that \( P \subset EXPTIME \), so it is possible that \( PSPACE \subset EXPTIME \).
For example: Instance checking:

\[ \mathcal{T}, \mathcal{A} \models a : C \iff \mathcal{T} \cup \mathcal{A} \cup \{ \text{not } a : C \} \text{ inconsistent} \]

Complexity of deciding ABox consistency may be harder than TBox reasoning, but it usually is not. For \( \mathcal{ALC} \) it is PSPACE/EXPTIME complete.