E. Description Logics

This section is based on material from

- Ian Horrocks: http://www.cs.man.ac.uk/~horrocks/Teaching/cs646/

Description Logics

- OWL DL ist äquivalent zur Beschreibungslogik $\mathcal{SHOIN}(D_n)$. Auf letzterer basiert also die Semantik von OWL DL.
- Unter Beschreibungslogiken (Description Logics) versteht man eine Familie von Teilsprachen der Prädikatenlogik 1. Stufe, die entscheidbar sind.
- $\mathcal{SHOIN}(D_n)$ ist eine relativ komplexe Beschreibungslogik.
- Um einen ersten Einblick in das Prinzip der Beschreibungslogiken zu erhalten, werfen wir zum Abschluss der Vorlesung einen Blick auf etwas abgespeckte Fassungen.

Literatur:


Aside: Semantics and Model Theories

- Ontology/KR languages aim to model (part of) world
- Terms in language correspond to entities in world
- Meaning given by, e.g.:
  - Mapping to another formalism, such as FOL, with own well defined semantics
  - or a bespoke Model Theory (MT)
- MT defines relationship between syntax and interpretations
  - Can be many interpretations (models) of one piece of syntax
  - Models supposed to be analogue of (part of) world
    - E.g., elements of model correspond to objects in world
  - Formal relationship between syntax and models
    - Structure of models reflect relationships specified in syntax
    - Inference (e.g., subsumption) defined in terms of MT
      - E.g., $\mathcal{T} \models A \subseteq B$ iff in every model of $\mathcal{T}$, $\text{ext}(A) \subseteq \text{ext}(B)$

Aside: Set Based Model Theory

- Many logics (including standard First Order Logic) use a model theory based on Zermelo-Frankel set theory
- The domain of discourse (i.e., the part of the world being modelled) is represented as a set (often referred as $\Delta$)
- Objects in the world are interpreted as elements of $\Delta$
  - Classes/concepts (unary predicates) are subsets of $\Delta$
  - Properties/roles (binary predicates) are subsets of $\Delta \times \Delta$ (i.e., $\Delta^2$)
  - Ternary predicates are subsets of $\Delta^3$ etc.
- The sub-class relationship between classes can be interpreted as set inclusion
- Doesn’t work for RDF, because in RDF a class (set) can be a member (element) of another class (set)
  - In Z-F set theory, elements of classes are atomic (no structure)
Aside: Set Based Model Theory Example

World

Model

Interpretation

Daisy isA Cow
Cow kindOf Animal
Mary isA Person
Person kindOf Animal
Z123ABC isA Car
Mary drives Z123ABC

\{a,b,\ldots\} \subseteq \Delta \times \Delta

What Are Description Logics?

- A family of logic based Knowledge Representation formalisms
  - Descendants of semantic networks and KL-ONE
  - Describe domain in terms of concepts (classes), roles (relationships) and individuals
- Distinguished by:
  - Formal semantics (typically model theoretic)
    - Decidable fragments of FOL
    - Closely related to Propositional Modal & Dynamic Logics
  - Provision of inference services
    - Sound and complete decision procedures for key problems
    - Implemented systems (highly optimised)

Aside: Set Based Model Theory Example

- Formally, the vocabulary is the set of names we use in our model of (part of) the world
  - \{Daisy, Cow, Animal, Mary, Person, Z123ABC, Car, drives, \ldots\}
- An interpretation \( \mathcal{I} \) is a tuple \( \langle \Delta, \mathcal{I} \rangle \)
  - \( \Delta \) is the domain (a set)
  - \( \mathcal{I} \) is a mapping that maps
    - Names of objects to elements of \( \Delta \)
    - Names of unary predicates (classes/concepts) to subsets of \( \Delta \)
    - Names of binary predicates (properties/roles) to subsets of \( \Delta \times \Delta \)
    - And so on for higher arity predicates (if any)

DL Architecture

Knowledge Base

Tbox (schema)

\textbf{Man} = \textbf{Human} \sqcap \textbf{Male}
\textbf{Happy-Father} = \textbf{Man} \sqcap \exists \textbf{has-child}
\textbf{Female} \sqcap \ldots

Abox (data)

John : \textbf{Happy-Father}
\langle John, Mary \rangle : \textbf{has-child}
Short History of Description Logics

Phase 1:
- Incomplete systems (Back, Classic, Loom, . . . )
- Based on structural algorithms

Phase 2:
- Development of tableau algorithms and complexity results
- Tableau-based systems for Pspace logics (e.g., Kris, Crack)
- Investigation of optimisation techniques

Phase 3:
- Tableau algorithms for very expressive DLs
- Highly optimised tableau systems for ExpTime logics (e.g., FaCT, DLP, Racer)
- Relationship to modal logic and decidable fragments of FOL

Latest Developments

Phase 4:
- Mature implementations
- Mainstream applications and Tools
  - Databases
    - Consistency of conceptual schemata (EER, UML etc.)
    - Schema integration
    - Query subsumption (w.r.t. a conceptual schema)
  - Ontologies and Semantic Web (and Grid)
    - Ontology engineering (design, maintenance, integration)
    - Reasoning with ontology-based markup (meta-data)
    - Service description and discovery
  - Commercial implementations
    - Cerebra system from Network Inference Ltd

From RDF to OWL

- Two languages developed to satisfy the requirements
  - OIL: developed by group of (largely) European researchers (several from EU OntoKnowledge project)
  - DAML-ONT: developed by group of (largely) US researchers (in DARPA DAML programme)
- Efforts merged to produce DAML+OIL
  - Development was carried out by “Joint EU/US Committee on Agent Markup Languages”
  - Extends (“DL subset” of) RDF
- DAML+OIL submitted to W3C as basis for standardisation
  - Web-Ontology (WebOnt) Working Group formed
  - WebOnt group developed OWL language based on DAML+OIL
  - OWL language now a W3C Recommendation (i.e., a standard like HTML and XML)

Description Logic Family

- DLs are a family of logic based KR formalisms
- Particular languages mainly characterised by:
  - Set of constructors for building complex concepts and roles from simpler ones
  - Set of axioms for asserting facts about concepts, roles and individuals
- $\mathcal{ALC}$ is the smallest DL that is propositionally closed
  - Constructors include booleans (and, or, not), and
  - Restrictions on role successors
  - E.g., concept describing “happy fathers” could be written:
    \[
    \text{Man} \land \exists \text{hasChild.Female} \land \exists \text{hasChild.Male} \land \forall \text{hasChild.(Rich} \lor \text{Happy)}
    \]
**DL Concept and Role Constructors**

- Range of other constructors found in DLs, including:
  - Number restrictions (cardinality constraints) on roles, e.g., \( \geq 3 \) hasChild, \( \leq 1 \) hasMother
  - Qualified number restrictions, e.g., \( \geq 2 \) hasChild.Female, \( \leq 1 \) hasParent.Male
  - Nominals (singleton concepts), e.g., \{Italy\}
  - Concrete domains (datatypes), e.g., hasAge.(\( \leq 21 \))
  - Inverse roles, e.g., hasChild\( \sim \) (hasParent)
  - Transitive roles, e.g., hasChild* (descendant)
  - Role composition, e.g., hasParent \( \circ \) hasBrother (uncle)

**DL Knowledge Base**

- DL Knowledge Base (KB) normally separated into 2 parts:
  - TBox is a set of axioms describing structure of domain (i.e., a conceptual schema), e.g.:
    - HappyFather = Man \( \land \) \( \exists \)hasChild.Female \( \land \)...
    - Elephant = Animal \( \land \) Large \( \land \) Grey
    - transitive(ancestor)
  - ABox is a set of axioms describing a concrete situation (data), e.g.:
    - John:HappyFather
    - \(<John,Mary>\):hasChild
  - Separation has no logical significance
    - But may be conceptually and implementationally convenient

**OWL as DL: Class Constructors**

<table>
<thead>
<tr>
<th>Constructor</th>
<th>DL Syntax</th>
<th>Example</th>
<th>FOL Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>( C_1 \cap \ldots \cap C_n )</td>
<td>Human ( \cap ) Male</td>
<td>( C_1(x) \land \ldots \land C_n(x) )</td>
</tr>
<tr>
<td>unionOf</td>
<td>( C_1 \cup \ldots \cup C_n )</td>
<td>Doctor ( \cup ) Lawyer</td>
<td>( C_1(x) \lor \ldots \lor C_n(x) )</td>
</tr>
<tr>
<td>complementOf</td>
<td>( \neg C )</td>
<td>( \neg )Male</td>
<td>( \neg C(x) )</td>
</tr>
<tr>
<td>oneOf</td>
<td>( {x_1 } \cup \ldots \cup {x_n } )</td>
<td>( {john} \cup {mary} )</td>
<td>( \forall y. P(x, y) \rightarrow C(y) )</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>( \forall P.C )</td>
<td>( \forall )hasChild.Lawyer</td>
<td>( \forall y. P(x, y) \land C(y) )</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>( \exists P.C )</td>
<td>( \exists )hasChild.Male</td>
<td>( \exists y. P(x, y) )</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>( \leq n P )</td>
<td>( \leq 1 )hasChild</td>
<td>( \exists y. P(x, y) )</td>
</tr>
<tr>
<td>minCardinality</td>
<td>( \geq n P )</td>
<td>( \geq 2 )hasChild</td>
<td>( \exists y. P(x, y) )</td>
</tr>
</tbody>
</table>

**RDFS Syntax**

E.g., Person \( \cap \forall \)hasChild.Doctor \( \cap \exists \)hasChild.Doctor:

```xml
<owl:Class>
  <owl:intersectionOf rdf:parseType="collection">
    <owl:Class rdf:about="#Person"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:toClass>
        <owl:unionOf rdf:parseType="collection">
          <owl:Class rdf:about="#Doctor"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild"/>
            <owl:hasClass rdf:resource="#Doctor"/>
          </owl:Restriction>
        </owl:unionOf>
      </owl:toClass>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```
OWL as DL: Axioms

- Axioms (mostly) reducible to inclusion ($\sqsubseteq$)
  - $C \equiv D$ iff both $C \sqsubseteq D$ and $D \sqsubseteq C$

- Obvious FOL equivalences
  - E.g., $C \equiv D$ iff $\forall x. C(x) \iff D(x)$,
  - $C \sqsubseteq D$ iff $\forall x. C(x) \Rightarrow D(x)$

XML Schema Datatypes in OWL

- OWL supports XML Schema primitive datatypes
  - E.g., integer, real, string, ...
- Strict separation between “object” classes and datatypes
  - Disjoint interpretation domain $\Delta_D$ for datatypes
    - For a datavalue $d$ holds $d^I \subseteq \Delta_D$
    - and $\Delta_D \cap \Delta^I = \emptyset$
  - Disjoint “object” and datatype properties
    - For a datatype property $P$ holds $P^I \subseteq \Delta^I \times \Delta_D$
    - For object property $S$ and datatype property $P$ hold $S^I \cap P^I = \emptyset$
- Equivalent to the “$(\mathbb{I})$” in $SHOIN(\mathbb{I})$

Why Separate Classes and Datatypes?

- Philosophical reasons:
  - Datatypes structured by built-in predicates
  - Not appropriate to form new datatypes using ontology language
- Practical reasons:
  - Ontology language remains simple and compact
  - Semantic integrity of ontology language not compromised
  - Implementability not compromised — can use hybrid reasoner

OWL DL Semantics

- Mapping OWL to equivalent DL ($SHOIN(\mathbb{I})$):
  - Facilitates provision of reasoning services (using DL systems)
  - Provides well defined semantics
- DL semantics defined by interpretations: $\mathcal{I} = (\Delta^I, \mathcal{I})$, where
  - $\Delta^I$ is the domain (a non-empty set)
  - $\mathcal{I}$ is an interpretation function that maps:
    - Concept (class) name $A$ to subset $A^I$ of $\Delta^I$
    - Role (property) name $R$ to binary relation $R^I$ over $\Delta^I$
    - Individual name $i$ to element $i^I$ of $\Delta^I$
DL Semantics

- Interpretation function $\mathcal{I}$ extends to concept expressions in the obvious way, i.e.:

\[
\begin{align*}
(C \cap D)^\mathcal{I} &= C^\mathcal{I} \cap D^\mathcal{I} \\
(C \cup D)^\mathcal{I} &= C^\mathcal{I} \cup D^\mathcal{I} \\
(\neg C)^\mathcal{I} &= \Delta^\mathcal{I} \setminus C^\mathcal{I} \\
\{x\}^\mathcal{I} &= \{x^\mathcal{I}\} \\
\exists R.C)^\mathcal{I} &= \{x \mid \exists y.(x, y) \in R^\mathcal{I} \land y \in C^\mathcal{I}\} \\
(\forall R.C)^\mathcal{I} &= \{x \mid \forall y.(x, y) \in R^\mathcal{I} \Rightarrow y \in C^\mathcal{I}\} \\
(\leq n R)^\mathcal{I} &= \{x \mid \#\{y \mid \langle x, y \rangle \in R^\mathcal{I}\} \leq n\} \\
(\geq n R)^\mathcal{I} &= \{x \mid \#\{y \mid \langle x, y \rangle \in R^\mathcal{I}\} \geq n\}
\end{align*}
\]

DL Knowledge Bases (Ontologies)

- An OWL ontology maps to a DL Knowledge Base $\mathcal{K} = \langle T, A \rangle$

  - $T$ (Tbox) is a set of axioms of the form:
    - $C \subseteq D$ (concept inclusion)
    - $C \equiv D$ (concept equivalence)
    - $R \subseteq S$ (role inclusion)
    - $R \equiv S$ (role equivalence)
    - $R^+ \subseteq R$ (role transitivity)

  - $A$ (Abox) is a set of axioms of the form
    - $x \in D$ (concept instantiation)
    - $\langle x, y \rangle \in R$ (role instantiation)

- Two sorts of Tbox axioms often distinguished
  - "Definitions"
    - $C \subseteq D$ or $C \equiv D$ where $C$ is a concept name
  - General Concept Inclusion axioms (GCIs)
    - $C \subseteq D$ where $C$ in an arbitrary concept

Knowledge Base Semantics

- An interpretation $\mathcal{I}$ satisfies (models) an axiom $A$ ($\mathcal{I} \models A$):
  - $\mathcal{I} \models C \subseteq D$ iff $C^\mathcal{I} \subseteq D^\mathcal{I}$
  - $\mathcal{I} \models C \equiv D$ iff $C^\mathcal{I} = D^\mathcal{I}$
  - $\mathcal{I} \models R \subseteq S$ iff $R^\mathcal{I} \subseteq S^\mathcal{I}$
  - $\mathcal{I} \models R \equiv S$ iff $R^\mathcal{I} = S^\mathcal{I}$
  - $\mathcal{I} \models R^+ \subseteq R$ iff $(R^+)^\mathcal{I} \subseteq R^\mathcal{I}$
  - $\mathcal{I} \models x \in D$ iff $x^\mathcal{I} \in D^\mathcal{I}$
  - $\mathcal{I} \models \langle x, y \rangle \in R$ iff $(x^\mathcal{I}, y^\mathcal{I}) \in R^\mathcal{I}$

- $\mathcal{I}$ satisfies a Tbox $T$ ($\mathcal{I} \models T$) iff $\mathcal{I}$ satisfies every axiom $A$ in $T$

- $\mathcal{I}$ satisfies an Abox $A$ ($\mathcal{I} \models A$) iff $\mathcal{I}$ satisfies every axiom $A$ in $A$

- $\mathcal{I}$ satisfies an KB $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$) iff $\mathcal{I}$ satisfies both $T$ and $A$

Inference Tasks

- Knowledge is correct (captures intuitions)
  - $C$ subsumes $D$ w.r.t. $\mathcal{K}$ iff for every model $\mathcal{I}$ of $\mathcal{K}$, $C^\mathcal{I} \subseteq D^\mathcal{I}$

- Knowledge is minimally redundant (no unintended synonyms)
  - $C$ is equivalent to $D$ w.r.t. $\mathcal{K}$ iff for every model $\mathcal{I}$ of $\mathcal{K}$, $C^\mathcal{I} = D^\mathcal{I}$

- Knowledge is meaningful (classes can have instances)
  - $C$ is satisfiable w.r.t. $\mathcal{K}$ iff there exists some model $\mathcal{I}$ of $\mathcal{K}$ s.t. $C^\mathcal{I} \neq \emptyset$

- Querying knowledge
  - $x$ is an instance of $C$ w.r.t. $\mathcal{K}$ iff for every model $\mathcal{I}$ of $\mathcal{K}$, $x^\mathcal{I} \in C^\mathcal{I}$
  - $\langle x, y \rangle$ is an instance of $R$ w.r.t. $\mathcal{K}$ iff for, every model $\mathcal{I}$ of $\mathcal{K}$, $(x^\mathcal{I}, y^\mathcal{I}) \in R^\mathcal{I}$

- Knowledge base consistency
  - A KB $\mathcal{K}$ is consistent iff there exists some model $\mathcal{I}$ of $\mathcal{K}$
DL Reasoning

- Tableau algorithms used to test satisfiability (consistency)
- Try to build a tree-like model $I$ of the input concept $C$
- Decompose $C$ syntactically
  - Apply tableau expansion rules
  - Infer constraints on elements of model
- Tableau rules correspond to constructors in logic ($\cap, \cup$ etc)
  - Some rules are nondeterministic (e.g., $\cap$, $\leq$)
  - In practice, this means search
- Stop when no more rules applicable or clash occurs
  - Clash is an obvious contradiction, e.g., $A(x), \neg A(x)$
- Cycle check (blocking) may be needed for termination
- $C$ satisfiable iff rules can be applied such that a fully expanded clash free tree is constructed

Highly Optimised Implementation

- Naive implementation leads to effective non-termination
- Modern systems include MANY optimisations
- Optimised classification (compute partial ordering)
  - Use enhanced traversal (exploit information from previous tests)
  - Use structural information to select classification order
- Optimised subsumption testing (search for models)
  - Normalisation and simplification of concepts
  - Absorption (rewriting) of general axioms
  - Davis-Putnam style semantic branching search
  - Dependency directed backtracking
  - Caching of satisfiability results and (partial) models
  - Heuristic ordering of propositional and modal expansion

What it means
- All Margherita_pizzas (amongst other things)
  - Are Pizzas
  - have_topping some Tomato_topping
  - have_topping some Mozzarella_topping
  - & because they are Pizzas
    - have_base some Pizza_base