An Introduction to Description Logics

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)

DL Architecture

Knowledge Base
- Tbox (schema)
  - Man ∈ Human ∧ Male
  - Happy-Father ≡ Man ∈ ∃ has-child Female

Abox (data)
- John : Happy-Father
  - (John, Mary) : has-child

Inference System

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

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
- At is the smallest DL that is propositionally closed
  - Constructors include boolean (and, or, not), and
  - Restrictions on role successors
  - E.g., concept describing "happy fathers" could be written:
    - Man ∧ hasChild.Female ∨ hasChild.Male
    - ∀ hasChild.(Rich ≡ Happy)
DL Concept and Role Constructors

- Range of other constructors found in DLs, including:
  - Number restrictions (cardinality constraints) on roles, e.g., $\exists^2\text{hasChild}$, $\exists^1\text{hasMother}$
  - Qualified number restrictions, e.g., $\exists^2\text{hasChild.Female}$, $\exists^1\text{hasParent.Male}$
  - Concrete domains (datatypes), e.g., hasAge.(21), earns.spends.<
  - Inverse roles, e.g., hasChild-(hasParent)
  - Transitive roles, e.g., hasChild*(descendant)
  - Role composition, e.g., hasParent o 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 := $\text{hasChild.Female} \land \ldots$
    - Elephant $\subseteq \text{Animal} \land \text{Large} \land \text{Grey} \land \text{transitive(ancestor)}$
  - ABox is a set of axioms describing a concrete situation (data), e.g.:
    - John:HaveFather
    - <John,Mary>:hasChild

OWL as DL: Class Constructors

- XML datatypes as well as classes in $\forall P.C$ and $\exists P.C$
  - E.g., $\exists\text{hasAge.nonNegativeInteger}$
- Arbitrarily complex nesting of constructors
  - E.g., Person $\cap \forall \text{hasChild}.(\text{Doctor} \cup \exists \text{hasChild.Doctor})$

RDFS Syntax

E.g., $\text{Person} \cap \forall \text{hasChild}.(\text{Doctor} \cup \exists \text{hasChild.Doctor})$:

```
<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 ($\subseteq$)
  - If $\forall P$ and $\exists Q$, $\exists Q \subseteq \forall P$
- Obvious FOL equivalences
  - $\forall P \subseteq Q \iff \forall P \subseteq Q \land \forall Q \subseteq Q'$
- Equivalent to the "$\exists D$" in $\text{SHOIN}(Dn)$

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 datatype $\phi \notin \Delta A$
    - For a datatype property $\phi$ and $\exists \phi$ and $\forall \phi$, $\forall \phi \subseteq \forall \phi \land \forall \exists D$
- Equivalent to the "$\exists D$" in $\text{SHOIN}(Dn)$
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
  - Implementation not compromised — can use hybrid reasoner

Only need sound and complete decision procedure for:
- General Concept Inclusion axioms (GCIs)
- Not appropriate to form new datatypes using ontology

Implementability not compromised — can use hybrid reasoner

Semantic integrity of ontology language not compromised

Ontology language remains simple and compact

Datatypes structured by built-in predicates

DL Semantics

- Interpretation function \( \cdot I \) extends to concept expressions in the obvious way, i.e.:
  - \( (C \cap D)^I = C^I \cap D^I \)
  - \( (C \cup D)^I = C^I \cup D^I \)
  - \( \neg C^I = \Delta^I \setminus C^I \)
  - \( \{x\}^I = \{x\} \)
  - \( (\exists R.C)^I = \{x | \exists y.(x, y) \in R^I \land y \in C^I \} \)
  - \( (\forall R.C)^I = \{x | \forall y.(x, y) \in R^I \Rightarrow y \in C^I \} \)
  - \( (\leq n R)^I = \{x | \# \{y | (x, y) \in R^I \} \leq n \} \)
  - \( (\geq n R)^I = \{x | \# \{y | (x, y) \in R^I \} \geq n \} \)

Interpretation Example

\[ \Delta = \{ v, w, x, y, z \} \]
\[ A = (v, w, x, y, z) \]
\[ B = (v, w) \]
\[ R^I = \{(v, w), (v, x), (y, x), (x, z)\} \]
\[ I_A = \{ x \} \]
\[ I_B = \{ y \} \]
\[ I_R = \{ (v, w), (v, x), (y, x), (x, z) \} \]

Knowledge Base Semantics

- An interpretation \( I \) satisfies (models) an axiom \( A \) \( (I \models A) \):
  - \( I \models C \subseteq D \)
  - \( I \models C \equiv D \)
  - \( I \models R \subseteq S \)
  - \( I \models R \equiv S \)
  - \( I \models C \sqcap D \subseteq C \sqcup D \)
  - \( I \models x \in D \iff x^I \in \Delta^I \)
  - \( I \models (x, y) \in R^I \iff (x^I, y^I) \in R^I \)

- \( I \) satisfies a Tbox \( T \) \( (I \models T) \) iff \( I \) satisfies every axiom \( A \) in \( T \)
- \( I \) satisfies an Abox \( A \) \( (I \models A) \) iff \( I \) satisfies every axiom \( A \) in \( A \)
- \( I \) satisfies a KB \( K \) \( (I \models K) \) iff \( I \) satisfies both \( T \) and \( A \)
Multiple Models - v- Single Model

- DL KB doesn't define a single model, it is a set of constraints that define a set of possible models.
  - No constraints (empty KB) means any model is possible.
  - More constraints means fewer models.
  - Too many constraints may mean no possible model (inconsistent KB).
- In contrast, DBs (and frame/rule KR systems) make assumptions such that DB/KB defines a single model.
  - Unique name assumption
  - Closed world assumption
  - Domain consists only of individuals named in the DB/KB
  - Extensions are as small as possible.

Reasoning (e.g., querying) is
Expressively powerful
Multiple models:
- Expressively powerful
  - Boolean connectives, including ~ and \( \lor \)
  - Can capture incomplete information
    - E.g., using \( \mu \) and \( \gamma \)
  - Monotonic
    - Adding information preserves truth
  - Reasoning (e.g., querying) is hard/slow
  - Queries may give counter-intuitive results in some cases

Single model:
- Expressively weaker (in most respects)
  - No negation or disjunction
  - Can't capture incomplete information
  - Nonmonotonic
    - Adding information does not preserve truth
  - Reasoning (e.g., querying) is easy/fast
  - Queries may give counter-intuitive results in some cases

Example of Single Model

\[
\begin{align*}
\text{KB} &= \{a:C, b:D, c:C, d:E, b:C\} \\
\text{I} &= \{v, w, x, y, z\} \\
\Delta &= \{a, b, c, d\} \\
C &= \{a, b, c\} \\
D &= \{a, b, c, d\} \\
\text{A} &= \{v, w, x, y, z\} \\
\forall x, y : E[x, y] \Rightarrow E[y, x] \\
\end{align*}
\]

Example of Multiple Models

\[
\begin{align*}
\text{KB} &= \emptyset \\
\text{I} &= \{v, w, x, y, z\} \\
\Delta &= \{v, w, x, y, z\} \\
\text{KB} &= \{a:C, b:D, c:C, d:E\} \\
\text{I} &= \{v, w, x, y, z\} \\
\Delta &= \{v, w, x, y, z\} \\
\text{KB} &= \{a:C, b:D, c:C, d:E, b:C\} \\
\text{I} &= \{v, w, x, y, z\} \\
\Delta &= \{v, w, x, y, z\} \\
\end{align*}
\]

Inference Tasks

- Knowledge is correct (captures intuitions)
  - \( C \) subsumes \( D \) w.r.t. \( \mathcal{K} \) iff for every model \( I \) of \( \mathcal{K} \), \( C \subseteq D \).
- Knowledge is minimally redundant (no unintended synonyms)
  - \( C \) is equivalent to \( D \) w.r.t. \( \mathcal{K} \) iff for every model \( I \) of \( \mathcal{K} \), \( C = D \).
- Knowledge is meaningful (classes can have instances)
  - \( C \) is satisifiable w.r.t. \( \mathcal{K} \) iff there exists some model \( I \) of \( \mathcal{K} \) s.t. \( C \neq \emptyset \).
- Querying knowledge
  - \( x \) is an instance of \( C \) w.r.t. \( \mathcal{K} \) iff for every model \( I \) of \( \mathcal{K} \), \( x \in I(C) \).
  - \( \{x, y\} \) is an instance of \( R \) w.r.t. \( \mathcal{K} \) iff for every model \( I \) of \( \mathcal{K} \), \( \{x, y\} \in I(R) \).
- Knowledge base consistency
  - A KB \( \mathcal{K} \) is consistent iff there exists some model \( I \) of \( \mathcal{K} \).