ONTOLOGIES & DESCRIPTION LOGICS

Parcours IA - Représentation des connaissances

Meghyn Bienvenu (LaBRI - CNRS & Université de Bordeaux)
OWL
Standard of the World Wide Web Consortium (W3C)

- Original version: OWL 1 from 2004
- Current version: **OWL 2** from 2012 (introduces EL, QL, RL profiles)

Motivated by the **Semantic Web**

Based upon (highly expressive) description logics, but offers
- **extra features**, like annotations, comments, imports
- several different formats (serializations):
  - XML, Turtle (RDF), **Manchester syntax**

**Much more well known** than DLs outside academia
Building blocks of OWL

- **individuals**
- **classes** $\rightsquigarrow$ **concepts** in DLs
- **object properties** $\rightsquigarrow$ **roles** in DLs
- **data properties**: 
  - connect individuals to data values (integers, strings)
- **annotation properties**: 
  - used to annotate ontologies, classes, axioms, etc.
  - metadata, not used for reasoning

**Axioms**

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>OWL (Manchester syntax)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C \sqsubseteq D$</td>
<td>$C \text{ SubClassOf: } D$</td>
</tr>
<tr>
<td>$C \equiv D$</td>
<td>$C \text{ EquivalentTo: } D$</td>
</tr>
<tr>
<td>$C \sqsubseteq \neg D$</td>
<td>$C \text{ DisjointWith: } D$</td>
</tr>
<tr>
<td>$R \sqsubseteq S$</td>
<td>$R \text{ SubPropertyOf: } S$</td>
</tr>
</tbody>
</table>
## Constructors in OWL

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>OWL (Manchester syntax)</th>
</tr>
</thead>
<tbody>
<tr>
<td>¬C</td>
<td>not C</td>
</tr>
<tr>
<td>C ⊔ D</td>
<td>C or D</td>
</tr>
<tr>
<td>C ⊓ D</td>
<td>C and D</td>
</tr>
<tr>
<td>(\exists r. C)</td>
<td>r some C</td>
</tr>
<tr>
<td>(\forall r. C)</td>
<td>r only C</td>
</tr>
<tr>
<td>(\exists r.{a})</td>
<td>r value {a}</td>
</tr>
<tr>
<td>(\geq n r. C)</td>
<td>r min n C</td>
</tr>
<tr>
<td>(\leq n r. C)</td>
<td>r max n C</td>
</tr>
<tr>
<td>r(^{-})</td>
<td>inverse r</td>
</tr>
<tr>
<td>⊤</td>
<td>owl:Thing</td>
</tr>
<tr>
<td>⊥</td>
<td>owl:Nothing</td>
</tr>
</tbody>
</table>
Female and Scientist

Cat or Dog

Person and (not Parent)

teaches some (CompSciCourse and (offeredBy some FrenchUniv))

teaches only CompSciCourse

hasIngredient max 5 owl:Thing
ONTOLEGY CONSTRUCTION
No single ‘correct’ ontology for any domain
- many possible ways to model a given domain
- different people will model in different ways
- need to consider how it will be used

Generally not possible to fully automate ontology design
- needs domain expert, analysis of application needs

Guidelines / methodologies for ontology design
- present approach from ‘Ontology Development 101’ (Noy & McGuinness, 2001)
1) Determine **domain and scope of the ontology**
   - what is the **domain** that the ontology will cover?
   - how is the ontology going to be used?
   - **what types of questions** should the information in the ontology allow us to answer? (competency questions)
   - who will use and maintain the ontology?

   → help to **choose appropriate level of detail, ontology language**

2) Consider **reusing (parts of) existing ontologies**

3) Make a **list all of the important terms**
4) Define the classes and organize them into hierarchy
   · different approaches: top-down / bottom-up / mixed

5) Define properties and link them to the classes
   · add existential / universal / cardinality restrictions to classes

6) Define other characteristics of properties
   · subproperties, inverses
   · functionality, transitivity, ...
   · domain and range

7) Create individuals and assertions about them

Note: iterative, not linear process, will likely need revisit steps!
Ensure **concept hierarchy reflects subclass (‘is-a’) relationship**

- If $C$ is subclass of $D$, every member of $C$ must be a member of $D$
- don’t just organize into classes by association!
  - example: HockeyStick and Goalie are *not* subclasses of Hockey

**Multiple inheritance is allowed** (e.g. $C$ subclass of $D$, $E$, and $F$)

**‘Sibling’ classes** should have **same level of generality**

Keep **number of sibling classes reasonable** (roughly 2-12)

- avoid having just a single class on a given ‘level’
- if large number of sibling classes, see if it would make sense to group them into intermediate classes
MORE ON STEP 4: DEFINING AND ORGANIZING CLASSES

When to add a new class?
- a subclass usually has some further characteristics / restrictions / participates in different relationships than its superclasses

Class or individual?
- not always obvious, really depends on application

Add disjointness axioms where appropriate
- useful for debugging

Limiting the scope
- cannot describe every possible aspect of the domain
- don’t go into more detail than you need for your application
Run the reasoner often and fix problems right away
- if too many problems, reasoner may fail / very slow
- better to fix modelling errors early, else lose lots of time

Errors to look for:
- unsatisfiable concepts (marked in red in Protégé)
- unsatisfiable KB (error message, owl:Thing subclassOf owl:Nothing)
- any unexpected entailments

Take advantage of Protégé’s ‘explain inference’ facility (button marked ?) to help understand source of problems
- justifications = minimal sets of axioms / assertions that are sufficient to get the entailment