OWL, Patterns, & FOL
COMP62342

Sean Bechhofer
sean.bechhofer@manchester.ac.uk

Uli Sattler
uli.sattler@manchester.ac.uk
A reminder: quotations and citations

- **Citations** [4] inform us where you got an idea/approach/result/technique/term…from

- **Reference** its source when you *take* an idea/result/example/…

- **Quote marks** “…” inform us where you got a phrase/sentence/paragraph from

- **Quote** when you *take* a sentence & reference its source!
  …even if it’s *only* 1 sentence or a short poem on your mom’s birthday card!
So far, we have looked at
• operational knowledge of OWL (FHKB)
• KR in general, its roles
• KA and competency questions
• formalising knowledge
• the semantics of OWL
Today:

- Semantic left-overs from last week
- Deepen your semantics: OWL & FOL & …
- Design **Patterns** in OWL
  - local ones
  - partonomies
- Design **Principles** in OWL:
  - multi-dimensional modelling &
  - post-coordination
  - PIMPS - an upper level ontology
- **Automated reasoning** about OWL ontologies:
  - a tableau-based algorithm to make
  - …implicit knowledge explicit
  - …our know KR *actionable*
Left-overs from last week
OWL 2 Semantics: an interpretation satisfying … (2)

• An interpretation I satisfies an axiom $\alpha$ if
  • $\alpha = C \text{ SubClassOf: } D$ and $C^I \subseteq D^I$
  • $\alpha = C \text{ EquivalentTo: } D$ and $C^I = D^I$
  • $\alpha = P \text{ SubPropertyOf: } S$ and $P^I \subseteq S^I$
  • $\alpha = P \text{ EquivalentTo: } S$ and $P^I = S^I$
  • …
  • $\alpha = x \text{ Type: } C$ and $x^I \in C^I$
  • $\alpha = x \text{ R y }$ and $(x^I, y^I) \in R^I$

• I satisfies an ontology $O$ if I satisfies every axiom $A$ in $O$
  • If I satisfies $O$, we call I a model of $O$

• See how the axioms in $O$ constrain interpretations:
  ✓ the more axioms you add to $O$, the fewer models $O$ has
  • …they do/don’t hold/are(n’t) satisfied in an ontology
    • in contrast, a class expression $C$ describes a set $C^I$ in I
Draw & Match Models to Ontologies!

O1 = {}
O2 = \{a:C, b:D, c:C, d:C\}
O3 = \{a:C, b:D, c:C, b:C, d:E\}
O4 = \{a:C, b:D, c:C, b:C, d:E\}
\hspace{1em} D \text{ SubClassOf } C
O5 = \{a:C, b:D, c:C, b:C, d:E\}
\hspace{1em} a \text{ R d,}
\hspace{1em} D \text{ SubClassOf } C,
\hspace{1em} D \text{ SubClassOf C,}
\hspace{1em} D \text{ SubClassOf}
\hspace{1em} S \text{ some C}
O6 = \{a:C, b:D, c:C, b:C, d:E\}
\hspace{1em} a \text{ R d,}
\hspace{1em} D \text{ SubClassOf } C,
\hspace{1em} D \text{ SubClassOf C,}
\hspace{1em} C \text{ SubClassOf } R \text{ only C } \}

I_1:
\Delta = \{v, w, x, y, z\}
C_I = \{v, w, y\}
D_I = \{x, y\} \hspace{1em} E_I = \{\}
R_I = \{(v, w), (v, y)\}
S_I = \{\}
\hspace{1em} a_I = v \hspace{1em} b_I = x
\hspace{1em} c_I = w \hspace{1em} d_I = y

I_2:
\Delta = \{v, w, x, y, z\}
C_I = \{v, w, y\}
D_I = \{x, y\} \hspace{1em} E_I = \{y\}
R_I = \{(v, w), (v, y)\}
S_I = \{\}
\hspace{1em} a_I = v \hspace{1em} b_I = x
\hspace{1em} c_I = w \hspace{1em} d_I = y

I_3:
\Delta = \{v, w, x, y, z\}
C_I = \{x, v, w, y\}
D_I = \{x, y\} \hspace{1em} E_I = \{y\}
R_I = \{(v, w), (v, y)\}
S_I = \{\}
\hspace{1em} a_I = v \hspace{1em} b_I = x
\hspace{1em} c_I = w \hspace{1em} d_I = y

I_4:
\Delta = \{v, w, x, y, z\}
C_I = \{x, v, w, y\}
D_I = \{x, y\} \hspace{1em} E_I = \{y\}
R_I = \{(v, w), (v, y)\}
S_I = \{(x,x), (y,x)\}
\hspace{1em} a_I = v \hspace{1em} b_I = x
\hspace{1em} c_I = w \hspace{1em} d_I = y

From Last Week
Let O be an ontology, α an axiom, and A, B classes, b an individual name:

- **O is consistent** if there exists some model I of O
  - i.e., there is an interpretation that satisfies all axioms in O
  - i.e., O isn’t self contradictory

- **O entails** α (written $O \models \alpha$) if α is satisfied in all models of O
  - i.e., α is a consequence of the axioms in O

- **A is satisfiable** w.r.t. O if $O \not\models A$ SubClassOf Nothing
  - i.e., there is a model I of O with $A_I \neq \emptyset$

- **b is an instance of** A w.r.t. O (written $O \models b:A$) if $b_I \subseteq A_I$ in every model I of O

**Theorem:**
1. O is consistent iff $O \not\models$ Thing SubClassOf Nothing
2. A is satisfiable w.r.t. O iff $O \cup \{n:A\}$ is consistent (where n doesn’t occur in O)
3. b is an instance of A in O iff $O \cup \{b:\text{not}(A)\}$ is not consistent
4. O entails A SubClassOf B iff $O \cup \{n:A \text{ and not}(B)\}$ is inconsistent
Let O be an ontology, α an axiom, and A, B classes, b an individual name:

- O is **consistent** if there exists some model I of O
  - i.e., there is an interpretation that satisfies all axioms in O
  - i.e., O isn’t self contradictory
- O **entails** α (written $O \models \alpha$) if α is satisfied in all models of O
  - i.e., α is a consequence of the axioms in O
- A is **satisfiable** w.r.t. O if $O \not\models A \sqsubseteq \text{Nothing}$
  - i.e., there is a model I of O with $A^I \neq \{\}$
- b is an **instance of** A w.r.t. O if $b^I \subseteq A^I$ in every model I of O

- O is **coherent** if every class name that occurs in O is satisfiable w.r.t O
- **Classifying O** is a reasoning service consisting of
  1. testing whether O is consistent; if yes, then
  2. checking, for each pair A,B of class names in O plus Thing, Nothing
     $O \not\models A \sqsubseteq B$
  3. checking, for each individual name b and class name A in O, whether $O \not\models b : A$
    …and returning the result in a suitable form: O’s **inferred class hierarchy**
A side note: Necessary and Sufficient Conditions

- **Classes** can be described in terms of *necessary* and *sufficient* conditions.
  - This differs from some frame-based languages where we only have necessary conditions.

- **Necessary** conditions
  - *SubClassOf* axioms
  - C SubClassOf: D...any instance of C is also an instance of D

- **Necessary & Sufficient** conditions
  - *EquivalentTo* axioms
  - C EquivalentTo: D...any instance of C is also an instance of D and vice versa, any instance of D is also an instance of C

- Allows us to perform automated **recognition** of individuals, i.e. \( O \models b : C \)

---

If it looks like a duck and walks like a duck, then it’s a duck!
OWL and Other Formalisms:
First Order Logic
Object-Oriented Formalisms
OWL and First Order Logic

• in COMP60332, you have learned a lot about FOL
• most of OWL 2 (and OWL 1) is a **decidable fragment of FOL**:

  Translate an OWL ontology $\mathcal{O}$ into FOL using $t()$ as follows:

  $$t(\mathcal{O}) = \{ \forall x. t_x(C) \Rightarrow t_x(D) \mid C \text{ SubClassOf } D \in \mathcal{O} \} \cup \{ t_x(C)[x/a] \mid a : C \in \mathcal{O} \} \cup \{ r(a, b) \mid (a, b) : r \in \mathcal{O} \}$$

• …we assume that we have replaced each axiom $C$ EquivalentTo $D$ in $\mathcal{O}$ with $C$ SubClassOf $D$, $D$ SubClassOf $C$
• …what is $t_x(C)$?
OWL and First Order Logic

Here is the translation $t_x()$ from an OWL ontology into FOL formulae in one free variable

\[
\begin{align*}
    t_x(A) &= A(x), & t_y(A) &= A(y), \\
    t_x(\text{not } C) &= \neg t_x(C), & t_y(\text{not } C) &= \ldots, \\
    t_x(C \text{ and } D) &= t_x(C) \land t_x(D), & t_y(C \text{ and } D) &= \ldots, \\
    t_x(C \text{ or } D) &= \ldots, & t_y(C \text{ or } D) &= \ldots, \\
    t_x(r \text{ some } C) &= \exists y. r(x, y) \land t_y(C), & t_y(r \text{ some } C) &= \ldots, \\
    t_x(r \text{ only } C) &= \ldots, & t_y(r \text{ only } C) &= \ldots.
\end{align*}
\]

Exercise:
1. Fill in the blanks
2. Why is $t_x(C)$ a formula in 1 free variable?
3. translate O6 to FOL
4. …what do you know about the 2 variable fragment of FOL?

\[
O6 = \{ a:C, b:D, c:C, b:C, d:E \\
a R d, \\
D \text{ SubClassOf } C, \\
D \text{ SubClassOf } S \text{ some } C, \\
C \text{ SubClassOf } R \text{ only } C \}
\]
Object Oriented Formalisms

Many formalisms use an “object oriented model” with

- **Objects/Instances/Individuals**
  - Elements of the domain of discourse
  - e.g., “Bob”
  - Possibly allowing descriptions of classes

- **Types/Classes/Concepts**
  - to describe sets of objects sharing certain characteristics
  - e.g., “Person”

- **Relations/Properties/Roles**
  - Sets of pairs (tuples) of objects
  - e.g., “likes”

- Such languages are/can be:
  - Well understood
  - Well specified
  - (Relatively) easy to use
  - Amenable to machine processing
Object Oriented Formalisms

OWL can be said to be object-oriented:

- **Objects/Instances/Individuals**
  - Elements of the domain of discourse
  - e.g., “Bob”
  - Possibly allowing descriptions of classes

- **Types/Classes/Concepts**
  - to describe sets of objects sharing certain characteristics
  - e.g., “Person”

- **Relations/Properties/Roles**
  - Sets of pairs (tuples) of objects
  - e.g., “likes”

- **Axioms** represent background knowledge, constraints, definitions, …
- **Careful**: SubClassOf is similar to **inheritance** but different:
  - inheritance can usually be over-ridden
  - SubClassOf can’t
  - in OWL, ‘multiple inheritance’ is normal
Other KR systems

- Protégé can be said to provide a **frame-based view** of an OWL ontology:
  - it gathers axiom by the class/property names on their left

- DBs, frame-based or other KR systems may make assumptions:
  1. **Unique name assumption**
     - Different names are always interpreted as different elements
  2. **Closed domain assumption**
     - Domain consists only of elements named in the DB/KB
  3. **Minimal models**
     - Extensions are as small as possible
  4. **Closed world assumption**
     - What isn’t entailed by O isn’t true
  5. **Open world assumption**: an axiom can be such that
     - it’s entailed by O or
     - it’s negation is entailed by O or
     - none of the above

**Question:** which of these does
- OWL make?
- a SQL DB make?
**Other KR systems: Single Model -v- Multiple Model**

**Multiple models:**
- Expressively powerful
  - Boolean connectives, including **not, or**
- Can capture incomplete information
  - E.g., using **or, some**
- Monotonic: adding information preserves entailments
- Reasoning (e.g., querying) is often complex: e.g., reasoning by case
- 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
- Often non-monotonic: adding information may invalidate entailments
- Reasoning (e.g., querying) is often easy
- Queries may give counter-intuitive results in some cases
Complete details about OWL

- here, we have concentrated on some **core** features of OWL, e.g., no
  - domain, range axioms
  - SubPropertyOf, InverseOf
  - datatype properties
  - ...
- we expect you to look these up!

- OWL is defined via a **Structural Specification**
- [http://www.w3.org/TR/owl2-syntax/](http://www.w3.org/TR/owl2-syntax/)
- Defines language independently of concrete syntaxes
- Conceptual structure and abstract syntax
  - UML diagrams and functional-style syntax used to define the language
  - Mappings to concrete syntaxes then given.
- The structural specification provides the foundation for implementations (e.g. OWL API as discussed later)
OWL Resources

- The OWL Technical Documentation is all available online from the W3C site.

  http://www.w3.org/TR/owl2-overview/

  All the OWL documents are relevant; we recommend in particular the
  - Overview
  - Primer
  - Reference Guide and
  - Manchester Syntax Guide

- Our Ontogenesis Blog at
Today:
✓ Semantic left-overs from last week
✓ Deepen your semantics: OWL & FOL & …
  • Design **Patterns** in OWL
    • local ones
    • partonomies
  • Design **Principles** in OWL:
    • multi-dimensional modelling &
    • post-coordination
    • PIMPS - an upper level ontology
  • **Automated reasoning** about OWL ontologies:
    • a tableau-based algorithm to make
    • …implicit knowledge explicit
    • …our know KR actionable
Patterns of axioms

• An axiom pattern is
  • a recurring regularity in how axioms are used or appear within an ontology

• The most common is
  • atomic SubClassOf axioms, i.e. SubClassOf axioms with class names on both sides
  • … but they get much more complex than that

• Usually, we’re referring to syntactic patterns:
  • how axioms are written,
  • but remember “axioms” are inferred as well as written
Patterns and design patterns

- **Software Design Patterns** are
  - well accepted solutions for common issues met in software construction

- **Ontology Design Patterns** ODPs are the same:
  - well accepted solutions for common issues met in ontology construction
  - but ontology engineers have barely agreed on well accepted problems, let alone their solutions

- ODPs often depend on one’s philosophical stance … we’ll mostly talk about *patterns* as recurring regularities of asserted axioms
Coding style: term normalisation

- Is a sort of pattern...
- What we want is:
  - **Class** names:
    - singular nouns with
    - initial capital letter,
    - spaces via CamelCase
  - **Individual** names:
    - all lower case,
    - spaces indicated by _
  - **Property** names:
    - initial lower case letter,
    - spaces via CamelCase
    - usually start with “is” or “has”
- All classes and individuals have a label, creator, description annotation property
Term normalisation $\subseteq$ applied naming convention

- A naming convention determines
  - what words to use, in
  - which order and
  - what one does about symbols and acronyms

- Adopt one
  - for both labels and URI fragments
  - both for the URI fragment and for the label
- Having a label is a “good practice”

See [http://ontogenesis.knowledgeblog.org/948](http://ontogenesis.knowledgeblog.org/948) for an introduction
How good names help modelling

• The help understanding relationships between terms: for example,
  • Thigh, shin, foot and toe are not “leg”, but “leg part”
  • Slice of tomato, tomato sauce, and tomato puree are not “Tomato” but “Tomato based product”
  • Eggs, milk, honey are not meat or animal, but “Animal Product”
  • Vinegared Rice is not Sushi, but “part of Sushi” of “Sushi Ingredient”

• Card sorting and the three card trick can help you here
• More on this later when we talk about upper level ontologies
Types of axiom patterns

- **Domain modelling patterns**: How to organise the axioms describing a domain
  - Works both in the
    - large: the whole ontology – and in the
    - small: how to describe a class/type of sushi
- **Language patterns**: Used to
  - take advantage of language features or
  - work around something missing in a language
  - The latter are used in the former
A first Axiom Pattern: the Property Closure Pattern

Class: Nigiri

SubClassOf Sushi,

hasIngredient some VinegaredRice,

hasIngredient some Fish

• Does Nigiri contain rice?
• Does Nigiri contain fish?
• Does Nigiri contain beef?
A first Axiom Pattern: the **Property Closure Pattern**

**Class:** Nigiri

**SubClassOf** Sushi,

hasIngredient **some** VinegaredRice,

hasIngredient **some** Fish

Which of these interpretations is a model of the above axiom?
A first Axiom Pattern: the Property Closure Pattern

**Class**: Nigiri

**SubClassOf** Sushi,

hasIngredient some VinegaredRice,

hasIngredient **some** Fish,

hasIngredient **only** (Fish or VinegaredRice)

Use **property closure pattern**
to avoid unintended models!
OWL’s Open World Assumption (OWA)

- Unless we have ‘constrained’ something it may be possible
  - e.g., for Nigiri to have ingredients other than rice & fish
- This behaviour is as “open world assumption”
  - OWL makes OWA

For
- the answer to “Does Nigiri have beef as ingredient” is “Maybe/Don’t know”

```
Class: Nigiri
  SubClassOf Sushi,
  hasIngredient some VinegaredRice,
  hasIngredient some Fish
```

```
DisjointClasses: VinegaredRice, Fish, Beef
Class: Nigiri
  SubClassOf Sushi,
  hasIngredient some VinegaredRice,
  hasIngredient some Fish,
  hasIngredient only (Fish or VinegaredRice)
```

- For
  - the answer to “Does Nigiri have beef as ingredient” is “No”!
A first Axiom Pattern: the **Property Closure Pattern**

- In summary, the property closure pattern for a property P is of the form

  Class: A

  SubClassOf ...
  
  P some B1,
  .... ,
  P some Bn,
  P only (B1 or … or Bn)
A second Axiom Pattern: the **Covering Axiom Pattern**

- Say we have Class X with subclasses Yi
  - e.g., UG, MSc, MRes, PhD are all subclasses of Student

- Now we *may* want to say that “any individual of class X has to be an individual of some class Yi”
  - i.e., class X is *covered by* classes Y1,…,Yk
  - e.g., every Student is

- To ensure this **coverage of X** by Y1,…Yk, we use the **covering axiom**:

  \[
  \text{Class: } Y_1 \text{ SubClassOf } X \\
  \text{Class: } Y_2 \text{ SubClassOf } X \\
  \vdots \\
  \text{Class: } Y_k \text{ SubClassOf } X \\
  \text{Class: } X \text{ SubClassOf: } (Y_1 \text{ or … or } Y_k)
  \]

- Quick exercise: translate the above axioms into FOL!
More information on closing patterns…. 

- [http://ontogenesis.knowledgeblog.org/1001](http://ontogenesis.knowledgeblog.org/1001)
- Lots of short, accessible articles about ontology stuff
A third Axiom Pattern: the (Value) **Partitions Pattern**

- Say we have Class X with subclasses Yi
  - e.g., UG, MSc, MRes, PhD are all subclasses of Student
- Now we *may* want to say that
  “no individual can be an instance 2 or more of these class Yi”
- How do we “partition” values **for properties** such as Size, Spicyness, etc:
  - E.g., we want to say that a person’s “Size”
    - must be one of the subclasses of Size and
    - only one of those sizes – and that
    - an individual size cannot be two kinds of size at the same time
A third Axiom Pattern: the (Value) **Partitions Pattern**

- **Class:** Small $\textbf{SubClassOf}$ Size
- **Class:** Medium $\textbf{SubClassOf}$ Size
- **Class:** Large $\textbf{SubClassOf}$ Size
- **DisjointClasses:** Small, Medium, Large
- **Class:** Size $\textbf{SubClassOf}$ (Medium or Small or Large)

Diagram:
- Size
- Small
- Medium
- Large
A fourth Axiom Pattern: the Entity Property Quality Pattern

Class: Small SubClassOf Size
Class: Medium SubClassOf Size
Class: Large SubClassOf Size
DisjointClasses: Small, Medium, Large
Class: Size SubClassOf (Medium or Small or Large)
Property: hasSize Characteristics: Functional
Range: Size Domain: Mammal
Class: Human SubClassOf hasSize some Size
Class: Child SubClassOf Human and hasSize only Small
A fourth Axiom Pattern: the **Entity Property Quality Pattern**

- Used to model descriptive features of things
  - possibly together with a value partition
- **OWL elements:**
  - for each feature or quality such as size, weight, etc:
    - functional property, e.g., has_size and
    - class for its values, e.g., Size
    - link these by stating that the class is the **range** of the property
    - state to which classes these qualities apply
      - via the **domain** of the property and
      - where they are necessary
- Using classes allows to make subpartitions
  - may overlap
  - may be related to concrete sizes and datatype properties
    - e.g. very large, moderately large
- Have a look at
  - [http://www.w3.org/TR/swbp-specified-values/](http://www.w3.org/TR/swbp-specified-values/)
  - [http://ontogenesis.knowledgeblog.org/1499](http://ontogenesis.knowledgeblog.org/1499)
Beyond Axiom Patterns: Composition, Parts and Wholes
Composition or Aggregation

- Describing a **whole** object by means of its **parts**
  - treating complex things as a single object

- What are the primary composition relationships?
- What inferences can we make?
- What might we have in our representation languages to support this?

- **Mereonomy** is the study of **parts**, **wholes**, and their relations

http://www.flickr.com/photos/hartini/2429653007/
Parts & wholes: Some examples

- Bristles are part of a toothbrush
- Wheels are part of a shopping trolley
- A car is partly iron
- Milk is part of a cappuccino
- A meter is part of a kilometer
- Manchester is part of England
- A tree is part of a forest
- A slice of pie is part of the pie
- A book chapter is part of a book
- I am part of the University of Manchester

These are different kinds of composition, with different characteristics and properties.
Confusing them may result in incorrect (or undesirable) inferences.
Properties of Composition

- [Winston, Chaffin, Herrmann 1987] and [Odell 1998] identify core properties:
  
  - **functional:**
    - Does the part bear a functional or structural relationship to the whole?
    - e.g., engine-car, wheel-bicycle
  
  - **homeomerous:**
    - Is the part the same kind of thing as the whole?
    - e.g., the North-West of England, a slice of bread
  
  - **invariant:**
    - Can the part be separated from the whole?
    - e.g., a hair of me, the bell of my bicycle

  ...next, we discuss *natural* combinations of these that give rise to interesting *part-whole relations*

  ...and don’t confuse P-W-Rs with is-a/SubClassOf:
  - engine is part of car, but not ‘is-a’!
1. P-W-R: Component-Integral Object

- A configuration of parts within a whole
- Bristles - toothbrush
- Scene - film
- A particular arrangement (not just haphazard)
- If components cease to support the overall pattern then different associations may arise
  - Handle ripped from a door of the car.
    - No longer a part but a piece
2. P-W-R: Material-Object

- Parts can’t be removed
- Capuccino is partly milk
- Bread is partly flour

- Define what objects are made of.
- Component-Integral can be separated
  – Car without a door handle still a Car
- Material-Object can’t
  – Bread without flour not bread
3. P-W-R: Portion-Object

- Almost like Material-Object, but parts are the same kinds of thing as whole
  - Slice of bread is a portion of bread
  - Meter is part of a kilometer

- Selective inheritance of properties
  - Ingredients of bread are ingredients of slice of bread
    - But with different quantities

- Slice, helping, segment, lump, drop etc.
4. P-W-R: Place-Area

- Unlike Portion-Object, pieces cannot be removed
- Manchester part of England
- Peak part of a mountain
- Often between places and locations.
- Pieces similar in nature.
5. P-W-R: Member-Bunch

- No requirement for a particular structural or functional relationship
- Tree part of a Forest
- Employee part of the Union
- Ship part of a Fleet
- I am part of the University of Manchester
6. P-W-R: Member-Partnership

• An invariant form of Member-Bunch

• Stan Laurel is part of Laurel and Hardy
  • Fred and Ginger are a dancing couple

• Removal of member destroys the partnership
  – a different partnership may result
Summary of Odell’s Compositional Relationships

<table>
<thead>
<tr>
<th></th>
<th>Functional</th>
<th>Homeomeric</th>
<th>Separable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component-Integral</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Material-Object</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Portion-Object</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Place-Area</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Member-Bunch</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Member-Partnership</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Dont’ confuse P-W-Rs with Non Compositional Relationships such as

• Topological inclusion
  – I am in the lecture theatre

• Classification inclusion
  – Catch 22 is a Book
  – It’s an instance of Book, not a part of it, so not Member-Bunch

• Attribution
  – Properties of an object can be confused with composition
  – Height of a Lighthouse isn’t part of it

• Attachment
  – Earrings aren’t part of Ears
  – Toes are part of Feet
  – Sometimes attachments are parts, but not always

• Ownership
  – A bicycle has wheels
  – I have a bicycle
  – A lot of modelling is about making the right distinctions and thus helping to get the right relationships between individuals
So what?
Modelling these in OWL
Transitivity

- We might expect part-whole or composition relationships to behave transitively.
  - But this is generally only true with the same kind of composition.

- Engine part of the Car
- Pistons part of the Engine
  ➡ Pistons part of the Car

- Sean’s arm part of Sean
- Sean part of School of Computer Science
  ➡ Sean’s arm part of School of Computer Science
Transitivity

- We might expect part-whole or composition relationships to behave transitively.
  - But this is generally only true with the same kind of composition.

- Engine part of the Car
  - Pistons part of the Engine
  - Pistons part of the Car

- Sean’s arm part of Sean
  - Sean part of School of Computer Science
  - Sean’s arm part of School of Computer Science
Transitivity

- In partonomies, we may want to identify direct parts
  - Piston directPartOf Engine; Engine directPartOf Car
  - Piston is not directPartOf Car, but is a partOf Car
- I want to query for all the direct parts of the Car, but not the direct parts of its direct parts.
  - So directPartOf shouldn’t be transitive

- Solution: provide a transitive superproperty

<table>
<thead>
<tr>
<th>Property</th>
<th>Characteristics</th>
<th>SubPropertyOf</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPartOf</td>
<td>Transitive</td>
<td>isPartOf</td>
</tr>
<tr>
<td>isDirectPartOf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Queries can use the superproperty to query transitive closure
- Assertions use the direct part of relationship
- A standard ontology design pattern, sometimes referred to as transitive reduction.
Aside: Transitivity and Subproperties

- Transitive property $R$ is one s.t. for any $I$ model of $O$, any $x,y,z$ in $\Delta$:
  - if $(x,y) \in R^I$ and $(y,z) \in R^I$, then $(x,z) \in R^I$
  - A superproperty of a transitive property is not necessarily transitive
  - A subproperty of a transitive property is not necessarily transitive

```plaintext
Property: knows
Property: hasFriend
SubPropertyOf: knows
Characteristics: Transitive
Property: hasBestFriend
SubPropertyOf: hasFriend
```
Aside: A note on Inverses

- OWL allows us to define inverse relationships

- If $P$ is the inverse of $Q$ in $O$, then for any $I$ model of $O$, any $x, y$ in $\Delta$: $(x, y) \in P^I$ iff $(y, x) \in Q^I$

- Be careful about what you can infer about inverse relationships:

  ...are all engines part of cars?
  - does this ontology entail that Engine $\text{SubClassOf}$ (isPartOf $\text{some}$ Car)?
Composition

- Composition provides a mechanism for describing a (class of) object(s) in terms of its parts.

- By considering basic properties of this part-whole relationship, we can identify different kinds of relationship.

- The different relationships then help us in identifying when, for example, we can (or can’t) apply transitivity.

- Explicitly separating these in our representation can avoid incorrect/invalid inferences.
Today:
✓ Semantic left-overs from last week
✓ Deepen your semantics: OWL & FOL & …
✓ Design **Patterns** in OWL
  local ones
  partonomies
• Design **Principles** in OWL:
  • multi-dimensional modelling &
  • post-coordination
  • PIMPS - an upper level ontology
• **Automated reasoning** about OWL ontologies:
  • a tableau-based algorithm to make
  • …implicit knowledge explicit
  • …our know KR *actionable*
Ontology Normalisation

• An ontology covers different **kinds of things**
  • each kind can come with its (class) hierarchy!
  ➡ poly-hierarchies are the norm
  • “Harry Potter and the Philosopher’s stone” is a book, a
    • children’s book (readers!),
    • work of fiction (literature category!)
    • written in English (language!)
    • available in paperback (form of printing/binding)
• Poly-hierarchies allow knowledge to be captured and appropriately queried
• They are difficult to build by hand
  • do we have “EnglishChildFictionPaperback” or
    “EnglishChildPaperbackFiction” or….
• Essentially impossible to get right and maintain
  • combinatorial explosion of terms!
• We can use OWL and automated reasoners to do the work for us
• … but how does one manage this and get it right?
Example: tangled medicine

shoulder_catches_during_movement
shoulder_feels_like_it_will_slip_out_of_place
shoulder_joint_feels_like_it_may_slip_out_of_place
shoulder_joint_pain_better_after_rest
shoulder_joint_pain-causes Difficulty_lying_on_affected_side
shoulder_joint_pain-causing_inability_to_sleep
shoulder_joint_pain-difficult_to_localize
shoulder_joint_pain-feels_better_after_normal_movement
shoulder_joint_pain_first_appears_at_night
shoulder_joint_pain-improved_by_medication
shoulder_joint_pain-improves_during_exercise__returns_later
shoulder_joint_pain_incr_by_raising_arm_above_shoulder_level
shoulder_joint_pain_increased_by
shoulder_joint_pain_increased_by_lifting
shoulder_joint_pain_increased_by_moving_arm_across_chest
shoulder_joint_pain_increased_by_reaching_around_the_back
shoulder_joint_pain_relieved_by_putting_arm_over_head
shoulder_joint_pain_sudden_onset
shoulder_joint_pain-unrelenting
shoulder_joint_pain-worse_on_rising
shoulder_joint_pain-worsens_with_extended_activity
shoulder_joint_popping_sound_heard
shoulder_joint_suddenly_gives_way
shoulder_seems_out_of_place
shoulder_seems_out_of_place__recollection_of_the_event
shoulder_seems_out_of_place_recurrent
shoulder_seems_out_of_place_which_resolved
shoulder_suddenly_locked_up
Example: “tangled” ontology of amino acids
There are several *dimensions* of classification here

- Identifiable *dimensions* are:
  - **amino acids** themselves – they have **side chains**
  - the **size** of the amino acids side chain
  - the **charge** on the side chain
  - the **polarity** of the side chain
  - The **hydrophobicity** of the side chain
- We can *normalise* these into separate hierarchies then put them back together again
- Our goal is to put entities into separate *trees* all formed on the same basis
### Untangling 1: separate dimensions

<table>
<thead>
<tr>
<th>Charge</th>
<th>Polarity</th>
<th>Hydrophobicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative</td>
<td>Polar</td>
<td>Hydrophobic</td>
</tr>
<tr>
<td>Neutral</td>
<td>Nonpolar</td>
<td>Hydrophilic</td>
</tr>
<tr>
<td>Positive</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Amino Acids
- Alanine
- Arginine
- Asparagine
- Cysteine
- Glutamate
- Glutamine
- Glycine
- Histidine
- Isoleucine
- Leucine
- Lysine
- Methionine
- Phenylalanine
- Proline
- Serine
- Threonine
- Tryptophan
- Tyrosine
- Valine
Untangling 1: separate dimensions

- Each separate dimension includes the same kind of thing
- Within a dimension, we don’t mix
  - self-standing things, processes, modifiers (qualities)
  - our classification by, for instance, structure and then charge
- We do that compositionally via defined classes and the automated reasoners
Untangling 2: relate dimensions using properties

Class: AminoAcid
  SubClassOf: hasSize some Size,
              hasPolarity some Polar,
              hasCharge some Charge,
              hasHydrophobicity some hydrophobicity

Class: Lysine
  SubClassOf: AminoAcid,
              hasSize some Large,
              hasCharge some Positive,
              hasPolarity some Polar,
              hasHydrophobicity some Hydrophilic

Amino Acids
  - Alanine
  - Arginine
  - Asparagine
  - Cysteine
  - Glutamate

Polarity
  - Polar
  - Nonpolar

Size
  - Tiny
  - Small
  - Medium
  - Large

Charge
  - Negative
  - Neutral
  - Positive

Hydrophobicity
  - Hydrophobic
  - Hydrophilic
Untangling 3: Describe **relevant** terms

**Class**: LargeAminoAcid  
**EquivalentTo**: AminoAcid  
*and* hasSize **some** Large

**Class**: PositiveAminoAcid  
**EquivalentTo**: AminoAcid  
*and* hasCharge **some** Positive

**Class**: LargePositiveAminoAcid  
**EquivalentTo**: LargeAminoAcid and PositiveAminoAcid
Post-Coordination

- This poly-hierarchical/multi-dimensional modelling style in OWL allows us to use post-coordination
  - build **class expressions** and use them **like names**
  - i.e., we can ask a reasoner (via the OWL API)
    - for instances of (AminoAcid **and** (hasSize **some** Large) **and** (hasCharge **some** Positive))
    - whether (AminoAcid **and** (hasSize **some** Large) **and** (hasCharge **some** Neutral)) is satisfiable w.r.t O
  - relies on OWL reasoners/tools to be able to handle **class expressions** in the same way as they handle **class names**

- this saves us from having to give names to all combinations:
  - we can give names to some expressions
    - but we don’t have to
    - since the reasoner can **understand** expressions!
Patterns used

- The Amino acids ontology uses these five patterns:
  - Normalisation/Multidimensional modelling
  - EPQ
  - Closure (via it’s functional properties)
  - A covering axiom for all the amino acids
  - It’s own pattern for amino acids

- There is more information via
  - http://ontogenisis.knowledgeblog.org/tag/ontology-normalization
  - http://ontogenisis.knowledgeblog.org/1401
PIMPS - an Upper Level Ontologies
Upper Level Ontologies

- Domain neutral description of all entities
- Should be able to be used to describe any domain:
  - biology, art, politics, business, medicine, …
- The basic dimensions:
  - processes and the
  - things that participate in processes
- Different ULOs differ in
  - the ontology language they use
  - their level of detail
  - their view of the world
  - etc
- Much philosophical discussion
  - …been trying since 437 BCE and still not sorted it out
- So, we’ll do something simple: PIMPS
The PIMPS ontology in context
PIMPS: A Simple Domain Neutral Ontology

- Thing
  - Process
  - Immaterial
  - Material
  - Properties
    - Quality
    - Function
    - Role
    - Disposition
  - Sites
PIMPS: A Simple Domain Neutral Ontology

- **Process**
  - An entity that unfolds over time such that its identity changes
  - Not all of a process is present at a given time-point in that process
  - E.g., living, wedding, dying, eating, drinking, breathing, liberation, election…
  - Lots of “-ation” and “…ing” words

- **Material**
  - Self-standing things I can “hold in my hand”
  - E.g., ball, a car, a person, a leg, a pizza, a piece of seaweed, etc etc
  - All of it exists at any one point in time
  - All of Robert exists at any point in time, even though Robert himself actually changes
  - It retains its identity
PIMPS: A Simple Domain Neutral Ontology

• **Immaterial**
  • Self-standing things I can **not** “hold in my hand”
  • E.g., An idea, a goal, a wish, …
  • It exists at any one point in time
  • This idea may change over time but retains its identity

• **Properties**
  • Dependant (not-self-standing) things including
    • Quality, e.g. Size, Weight
    • Function, e.g., Control, Activation, Neutralisation
    • Role, e.g., Catalyst, Pathogen
    • Disposition, e.g., HeatResistance

• **Site**
  • point or area on a material entity
  • not to be confused with segments of that entity
Why use an upper level ontology?

• Consistent modelling style both within and between ontologies
• Primarily a guide to using properties consistently
• Continuants have parts that are continuants
• Processes have parts that are processes
• Independent continuants hasQuality some Quality and playRole some Role
• Independent continuant hasFunction some Function
• Independent continuants participate in processes
• Sites occupy some material entity
• Use property hierarchies…
Today:
✓ Semantic left-overs from last week
✓ Deepen your semantics: OWL & FOL & …
✓ Design **Patterns** in OWL
  local ones
  partonomies
Design **Principles** in OWL:
  multi-dimensional modelling &
  post-coordination
✓ PIMPS - an upper level ontology
  • **Automated reasoning** about OWL ontologies:
    • a tableau-based algorithm to make
    • …implicit knowledge explicit
    • …our know KR **actionable**