Assessed assignment

- **Assessed assignment 1**
  - Due 17 Feb.
  - 3 questions
  - Level 10 students answer Q1 and one other

- **Q1 Understand an OWL ontology**
  - Install Protégé and download the clothing.owl ontology from the KMM website
  - Answer parts i. to v. by editing (and saving) the ontology
    » Submit the revised ontology electronically
  - Each part i. to v. also requires a written answer
  - Questions relate to a line of clothing inspired by the Beatles Sgt. Pepper album cover

---

Manchester OWL Syntax

- **Derived from the OWL abstract syntax, but less verbose**
  - Aims to be easier to read and write
    » Especially for non-logicians
  - Minimal use of (
  - Allows DL expressions to be written in an English-like grammar, for email exchanges, GUIs etc

- **Previously...**
  - ALC / SHOIN / SHOIQ logical syntax: “VR.C”
  - OWL abstract syntax: `< Restriction>
    < onProperty R> < allValuesFrom C >>`
  - **Manchester syntax:** “R only C”

---

Assessed assignment

- **Q2 Describe an existing ontology**
  - E.g. one covered in lecture 6
  - 500 words / 1 page (750 words for level 10)
  - Summarise concisely
    » Include example concept definitions

- **Q3 Describe the work of Linnaeus**
  - 500 words / 1 page (750 words for level 10)

---

Manchester OWL Syntax

- **Observed that DL syntax is cryptic**
- **Quantifier Role. Concept order can be confusing and misread:**
  - Person \(\ni \exists\text{eats.Meat}\)
  - **correct:** Persons that eat (among other things) some Meat
  - **incorrect:** some Persons eat Meat
  - Manchester syntax is:
    - Person that eats some Meat \([\text{Person} \ni \exists\text{eats.Meat}]\)
    - Person that eats only Meat \([\text{Person} \ni \forall\text{eats.Meat}]\)
### Manchester OWL Syntax

<table>
<thead>
<tr>
<th>DL Syntax</th>
<th>Manchester Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>\neg C</td>
<td>not C</td>
<td>not Male</td>
</tr>
<tr>
<td>C \lor D</td>
<td>C or D</td>
<td>Man or Woman</td>
</tr>
<tr>
<td>C \land D</td>
<td>C and D</td>
<td>Parent and Man</td>
</tr>
<tr>
<td>\forall R.C</td>
<td>R only C</td>
<td>hasColleague only Professor</td>
</tr>
<tr>
<td>\exists R.C</td>
<td>R some C</td>
<td>hasColleague some Professor</td>
</tr>
<tr>
<td>\leq n R</td>
<td>R min n</td>
<td>hasColleague min 3</td>
</tr>
<tr>
<td>\geq n R</td>
<td>R max n</td>
<td>hasColleague max 3</td>
</tr>
<tr>
<td>= n R</td>
<td>R exactly 3</td>
<td>hasColleague exactly 3</td>
</tr>
<tr>
<td>\exists R.a</td>
<td>R value a</td>
<td>hasColleague value Fred</td>
</tr>
</tbody>
</table>

### In case of ambiguity of scope, a precedence order is defined (from highest to lowest):
- some, all, value, min, max, exactly, that
- not
- and
- or

**Example:**
Person that
(hasChild some (Person and (hasChild only Man) and (hasChild some Person))

### ‘onlysome’ design pattern
- Common to specify “eats some Meat and eats only Meat”
- The onlysome pattern makes this easier to state
E.g. Pizza that hasTopping onlysome [MozzarellaTopping, TomatoTopping]

**Example:**
(hasTopping some MozzarellaTopping) and
(hasTopping some TomatoTopping) and
(hasTopping only (MozzarellaTopping or TomatoTopping))

### Protégé 4 allows class expressions to be entered by typing the Manchester OWL
- E.g. define VegetarianPizza

**Example:**
Pizza and not (hasTopping some FishTopping)
and not (hasTopping some MeatTopping)

---

**Protégé 4 GUI**

1. **Manchester**
   - Pizza
   - and not (hasTopping some FishTopping)
   - and not (hasTopping some MeatTopping)

2. **Protégé 4**
   - Equivalent classes

---

### OWL and Protégé

**KMM ontology Lecture 5**
The original OWL has been extended
- OWL 1.1 and 2 are based on the SROIQ logic
- Adds new ways to reason about roles R
- Adds new cardinality constraints

These extensions are seen as useful in applications and technically feasible
- SRIOQ is decidable

New:
- Roles
- Number restrictions
- Proof
- Syntax

subPropertyOf

subPropertyOf(hasMother, hasParent)
subPropertyOf(P, Q): P(x, y) ⇒ Q(x, y)

In DL: hasMother ⊑ hasParent

Show: ∃ hasMother.Person ⊑ ∃ hasParent.Person

role inclusion: hasMother ⊑ hasParent

By role inclusion: hasMother ⊑ hasParent

Reasoning about roles increases expressivity

The set of roles is the set of role names, plus an inverse relation for each role name

Formally, let RN be the set of role names
the set of roles is RN ∪ {R′ | R ∈ RN}
where R′ is the inverse role of R

R′ ⊆ A∗ A
(R′) = {<y,x> | <x,y> ∈ R}

The function Inv() applies to roles:
Inv(R) = R′      Inv(R′) = R
**The Role box R includes**
- The role hierarchy
- Role inclusion axioms e.g. owns ⊑ hasPart ⊑ owns
- Role assertions

**Role inclusion axioms have some restrictions that prevent cyclic dependencies, these are valid:**

\[ R \circ S \subseteq R; \quad S \circ R \subseteq R; \quad R \circ R \subseteq R; \quad S \subseteq S \]

More generally, \( w \subseteq R \) iff \( \text{Inv}(w) \subseteq \text{Inv}(R) \)
where \( w \) is a string of role names

---

**Manchester OWL Syntax**

**Manchester syntax has been extended to OWL 2**
- Property chains
  partOf or partOf
  is written: partOf o partOf  i.e. with an small letter o

---

**OWL 2: Roles**

- Role assertions
  Sym(R) if \( \langle x,y \rangle \in R \implies \langle y,x \rangle \in R' \)
  Tra(R) if \( \langle x,y \rangle \in R \text{ and } \langle y,z \rangle \in R' \implies \langle x,z \rangle \in R' \)
  Ref(R) if \( \{ \langle x,x \rangle | x \in A \} \subseteq R' \)
  Irr(R) if \( R \cap \{ \langle x,x \rangle | x \in A \} = \emptyset \)
  Dis(R,S) if \( R \cap S = \emptyset \)

**In fact:**
Sym(R) = R \( \subseteq R \)
Tra(R) = R o R \( \subseteq R \)
So these role assertions are equivalent to inclusion axioms

---

**Number restrictions in OWL-DL**
- Minimum cardinality:
  \( \geq n \ R \) :: \( \{ x | x \in A \mid \#(\langle x,y \rangle \in R) \geq n \} \)
  E.g. The set of things with at least 2 parts-that-are-Wheels:
  \( \geq 2 \) hasWheel

**Number restrictions in OWL 2**
- Minimum cardinality specifies the class C for the n instances:
  \( \geq n \ R \cdot C \) :: \( \{ x | x \in A \mid \#(\langle x,y \rangle \in R \wedge y \in C) \geq n \} \)
  E.g. The set of things with at least 2 Wheels as parts:
  \( \geq 2 \) hasPart. Wheel

Similarly for maximum cardinality: \( \leq n \ R \cdot C \)
**Simple roles only** cannot say Tra(R) and \( \geq n \ R \cdot C \)

---

**OWL 2: Qualified number restrictions**

**Simple roles only** cannot say Tra(R) and \( \geq n \ R \cdot C \)

---

**Number restrictions in OWL-DL**
- Minimum cardinality:
  \( \geq n \ R \) :: \( \{ x | x \in A \mid \#(\langle x,y \rangle \in R) \geq n \} \)
  E.g. The set of things with at least 2 parts-that-are-Wheels:
  \( \geq 2 \) hasWheel

**Number restrictions in OWL 2**
- Minimum cardinality specifies the class C for the n instances:
  \( \geq n \ R \cdot C \) :: \( \{ x | x \in A \mid \#(\langle x,y \rangle \in R \wedge y \in C) \geq n \} \)
  E.g. The set of things with at least 2 Wheels as parts:
  \( \geq 2 \) hasPart. Wheel

Similarly for maximum cardinality: \( \leq n \ R \cdot C \)
**Simple roles only** cannot say Tra(R) and \( \geq n \ R \cdot C \)
Local reflexivity: \( \exists R.\text{Self} \)
e.g. \( \exists \text{likes.}\text{Self} \)
\[ ( \exists R.\text{Self} ) = \{ x | <x,x> \in R \} \]

Datatypes
- dataOneOf \{ set \} defines an enumerated datatype
- dataComplementOf (data range) returns the complement of the data range
- Datatype restriction uses datatype facet (from XML Schema)

Annotations
- Comments can be associated with subClassOf and axiom assertions

As with ALC tableaux, goals are constructed and translated into negation normal form

Additional equivalences:
\[ \neg(\exists n R.C) = (\forall(n+1) R.C) \]
\[ \neg(\forall(n+1) R.C) = (\exists n R.C) \]
\[ \neg(\exists 0 R.C) = \bot \]

Reasoning about role hierarchies and axioms increases the number of tableau rules
- Recall there were only 4 rules for ALC, one for each operator
- SROIQ has 18 rules
- Automata theory is used to deal with role inclusion
- Tableaux algorithm remains sound and complete
  » Subsumption is reduced to unsatisfiability:
    \[ C \subseteq D \text{ iff } C \cap \neg D \subseteq \bot \]
- Blocking is used to terminate the algorithm

By construction:
\( L(<a0, a1>) = \{ \text{partOf} \} \) \( L(<a1, a2>) = \{ \text{partOf} \} \)
By axiom \( \text{partOf} \circ \text{partOf} \subseteq \text{partOf} \)
\( \forall \text{partOf}. \neg C \) can be added to \( L(a1) \) and so
\( L(a2) = \{ C, \neg C \} \) showing a contradiction
**OWL 2 Reasoning**

- Number restrictions in OWL 2
  - The tableaux for SROIQ has generating rules and shrinking rules
    » “The even more irresistible SROIQ” Horrocks, I., Kutz, O. and Sattler, U. KR 2006

\[
\begin{array}{c}
\text{hasPart} \\
\text{Wheel} \\
\end{array}
\begin{array}{c}
\text{hasPart} \\
\text{Wheel} \\
\end{array}
\]

1. Eliminate $\exists_2$ to create 2 new nodes $a1 a2$
2. New rule for $\exists nR.C$ Clash if $> n$ nodes $y$, where $L(y)$ includes $C$

**OWL Functional Syntax**

- OWL 2 has a functional syntax and an XML syntax
  - XML syntax is not based on RDF/XML
  - XML schema is defined

**Functional Syntax Grammar**

Example 1: $\forall \text{partOf.Car}$
ObjectAllValuesFrom (http://www.inf.org#partOf http://www.inf.org#Car)

Example 2: $(\text{owns} \circ \text{hasPart}) \sqsubseteq \text{owns}$
SubObjectPropertyOf(
  SubObjectPropertyChain(http://www.inf.org#owns http://www.inf.org#hasPart)
  http://www.inf.org#owns)

**OWL XML Syntax**

Example 1: $\forall \text{partOf.Car}$
In XML:

```xml
<ObjectAllValuesFrom>
  <ObjectProperty IRI = "http://www.inf.org#partOf"/>
  <Class IRI = "http://www.inf.org#Car"/>
</ObjectAllValuesFrom>
```

Example 2: $(\text{owns} \circ \text{hasPart}) \sqsubseteq \text{owns}$
In XML:

```xml
<SubObjectPropertyOf>
  <ObjectPropertyChain>
    <ObjectProperty IRI = "http://www.inf.org#owns"/>
    <ObjectProperty IRI = "http://www.inf.org#hasPart"/>
  </ObjectPropertyChain>
  <ObjectProperty IRI = "http://www.inf.org#owns"/>
</SubObjectPropertyOf>
```

**OWL 2 Summary**

- OWL 2 extends OWL DL
- Adds the role box (hierarchy, assertions and inclusion axioms)
- Adds qualified number constraints
- Reasoning remains sounds and decidable
- XML syntax is based on a schema, not on RDF/XML