OWL, Patterns, & FOL

Yizheng Zhao
zhaoyz@nju.edu.cn

Next:

- 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*

OWL 2 Semantics: an interpretation satisfying … (2)

From Last Week

- An interpretation I **satisfies an axiom α** if
  - α = C SubClassOf: D and $C^I \subseteq D^I$
  - α = C EquivalentTo: D and $C^I = D^I$
  - α = P SubPropertyOf: S and $P^I \subseteq S^I$
  - α = P EquivalentTo: S and $P^I = S^I$
  - …
  - α = x Type: C and $x^I \in C^I$
  - α = x R y and $(x^I, y^I) \in R^I$

- I **satisfies an ontology O** if I satisfies every axiom α 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
D SubClassOf C}

O5 = {a:C, b:D, c:C, b:C, d:E
a R d,
D SubClassOf C,
D SubClassOf
S some C}

O6 = {a:C, b:D, c:C, b:C, d:E
a R d,
D SubClassOf C,
D SubClassOf
S some C,
C SubClassOf R only C }

I_1:
$\Delta = \{v, w, x, y, z\}$

$C^I = \{v, w, y\}$
$D^I = \{x, y\}$ $E^I = \{\}$

$R^I = \{(v, w), (v, y)\}$
$S^I = \{\}$

$a^I = v$ $b^I = x$
$c^I = w$ $d^I = y$

I_2:
$\Delta = \{v, w, x, y, z\}$

$C^I = \{v, w, y\}$
$D^I = \{x, y\}$ $E^I = \{y\}$

$R^I = \{(v, w), (v, y)\}$
$S^I = \{\}$

$a^I = v$ $b^I = x$
$c^I = w$ $d^I = y$

I_3:
$\Delta = \{v, w, x, y, z\}$

$C^I = \{x, v, w, y\}$
$D^I = \{x, y\}$ $E^I = \{y\}$

$R^I = \{(v, w), (v, y)\}$
$S^I = \{\}$

$a^I = v$ $b^I = x$
$c^I = w$ $d^I = y$

I_4:
$\Delta = \{v, w, x, y, z\}$

$C^I = \{x, v, w, y\}$
$D^I = \{x, y\}$ $E^I = \{y\}$

$R^I = \{(v, w), (v, y)\}$
$S^I = \{(x,x), (y,x)\}$

$a^I = v$ $b^I = x$
$c^I = w$ $d^I = y$

OWL 2 Semantics: Entailments etc. (3)

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 $\models$ α) 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 $A^I \neq \{\}$
- 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:not(A)} is not consistent
4. O entails A SubClassOf B iff O $\cup$ {n:A and not(B)} is inconsistent

OWL 2 Semantics: Entailments etc. (3) ctd

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 $\models$α) 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 $A^I \neq \{\}$
- b is an **instance of** A w.r.t. O if $b^I \subseteq A^I$ in every model I of 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 whether O $\models$A SubClassOf B
3. checking, for each individual name b and class name A in O, whether O $\models$b:A

…and returning the result in a suitable form: O's **inferred class hierarchy**

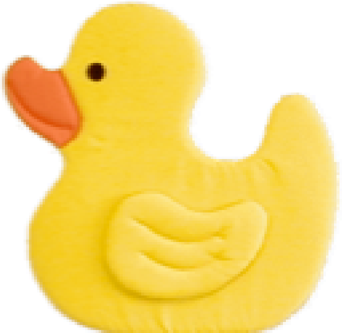
A side note: Necessary and Sufficient Conditions

- **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

- during your first year at NJU, you have learned a lot about FOL
- most of OWL 2 (and OWL 1) is a **decidable fragment of FOL:**

Translate an OWL ontology O into FOL using $t()$ as follows:

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

- …we assume that we have replaced each axiom C EquivalentTo D in 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

$t_x(A) = A(x),$	$t_y(A) = A(y),$
$t_x(\text{not C}) = \neg t_x(C),$	$t_y(\text{not C}) = \ldots,$
$t_x(C \text{ and D}) = t_x(C) \wedge t_x(D),$	$t_y(C \text{ and D}) = \ldots,$
$t_x(C \text{ or D}) = \ldots,$	$t_y(C \text{ or D}) = \ldots,$
$t_x(r \text{ some C}) = \exists y.r(x,y) \wedge t_y(C),$	$t_y(r \text{ some C}) = \ldots,$
$t_x(r \text{ only C}) = \ldots,$	$t_y(r \text{ only C}) = \ldots.$

Exercise:

1. Fill in the blanks
2. Why is tx(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 SubClassOf C,
D SubClassOf
S some C,
C SubClassOf R 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/
- 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
- http://www.sciencedirect.com/science/article/pii/S1570826808000413

Next:

✓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 in an ontology
- The most common is
  - atomic SubClassOf axioms,
    i.e. *A SubClassOf B* where A, B are class **names**
  - … but they get much more complex than that
- Usually, we're referring to **syntactic** patterns:
  - how axioms are written,
  - but remember "axioms" are entailed 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 similar:
  - 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

“Glucose transport” vs “transport of glucose”

- Adopt one
  - for both labels and URI fragments

- Having a label is a “good practice”

See 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”
  - Rice is not Sushi, but “part of Sushi” of “Sushi Ingredient”
- Card sorting and the three card trick can help you here

Types of axiom patterns

- **Naming Patterns**
  - see term normalisation, naming convention

- **Logical patterns** (also known as Language Patterns)
  axioms to
  - take advantage of language features or
  - work around something missing in a language

- **Content Patterns** (also known as Domain modelling patterns):
  axioms to describe certain phenoma/concepts in a domain
  - Works both in the
    - large: the whole ontology
    - small: how to describe a class/type of furniture