OWL, Patterns, & FOL
COMP62342

Sean Bechhofer
sean.bechhofer@manchester.ac.uk
Uli Sattler
uli.sattler@manchester.ac.uk

So far, we have looked at

- operational knowledge of OWL (Pizza)
- KR in general, its roles
- KA and competency questions
- formalising knowledge
- the semantics of OWL

Today:

- Deepen your semantics: OWL & FOL & …
- Design **Patterns** in OWL
  - local ones
  - partonomies
- Design **Principles** in OWL:
  - multi-dimensional modelling
  - PIMPS - an upper level ontology
  - post-coordination
- **Automated reasoning** about OWL ontologies:
  - a tableau-based algorithm to make
  - …implicit knowledge explicit
  - …our know KR *actionable*

Left-overs from last week:
More on OWL Semantics

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 $(x^I, y^I) \in R^I$

Check
OWL 2 Direct Semantics
for more!!!

- 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

OWL 2 Semantics: an interpretation satisfying … (2)

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

Check
OWL 2 Direct Semantics
for more!!!

- 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
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:
Δ = {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:
Δ = {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:
Δ = {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:
Δ = {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 $O \models \alpha$) 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 $A^I \neq \{\}$
- 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 $O \models \alpha$) 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 $A^I \neq \{\}$
- 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

  Constraints/Background knowledge
  - *SubClassOf* axioms
  - C SubClassOf: D…any instance of C is also an instance of D
- **Necessary & Sufficient** conditions

  Definitions
  - *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 ⊧ 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

MANCHESTER 1824

The University of Manchester

OWL and First Order Logic

- in COMP60332 or elsewhere, 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:

$$
\begin{aligned}
t(\mathcal{O}) = & \{\forall x.t_x(C) \Rightarrow t_x(D) \mid C \textsf{ SubClassOf D} \in \mathcal{O}\} \cup \\
& \{t_x(C)[x/a] \mid a\colon C \in \mathcal{O}\} \cup \\
& \{r(a,b) \mid (a,b)\colon r \in \mathcal{O}\}
\end{aligned}
$$

- …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

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

Exercise:

1. Fill in the blanks
2. Why is $t_x(C)$ a formula in 1 free variable?
3. Translate O6 to FOL
4. …have you heard 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

MANCHESTER 1824

The University of Manchester

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)

MANCHESTER 1824

The University of Manchester

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://ontogenesis.knowledgeblog.org/

Today:

✓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

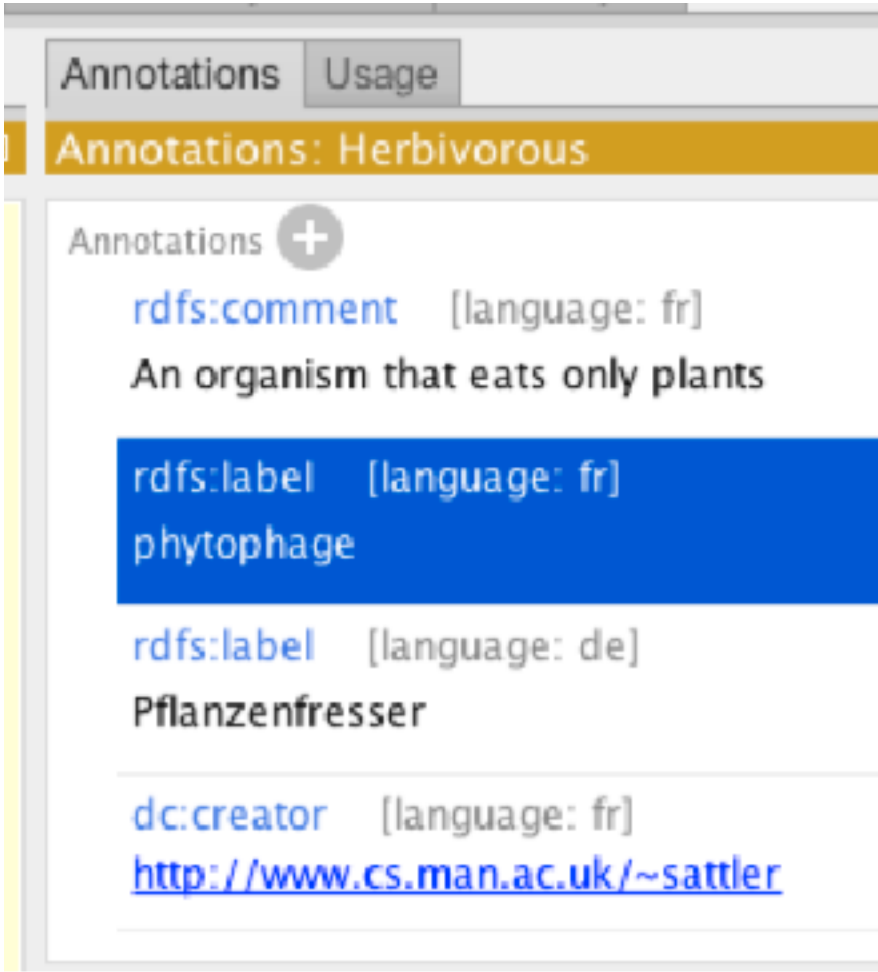
- An **axiom pattern** is
  - a recurring regularity in how axioms are used in an ontology
- The most common is
  - atomic SubClassOf axioms,
    i.e. *A SubClassOf B* where A, B are class **names**
  - … but they get much more complex than that
- Usually, we're referring to **syntactic** patterns:
  - 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**



MANCHESTER 1824

The University of Manchester

Term normalisation ⊆ 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
- Having a label is a “good practice”

“Glucose transport” vs “transport of glucose”

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”
  - Pizza base is not Pizza, but “part of Pizza” of “Pizza 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

1st Logical Pattern: the **Property Closure Pattern**

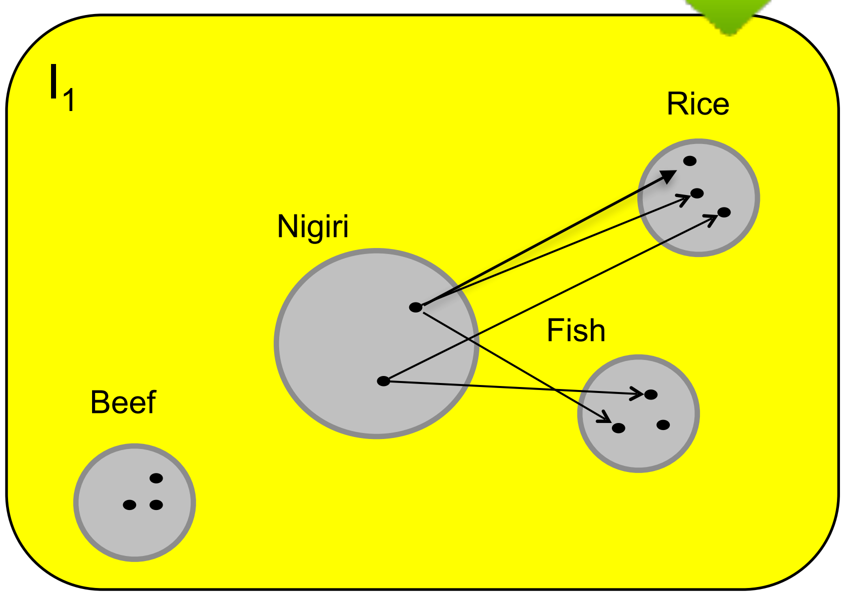
Class: Nigiri
SubClassOf Sushi,
hasIngredient **some** Rice,
hasIngredient **some** Fish

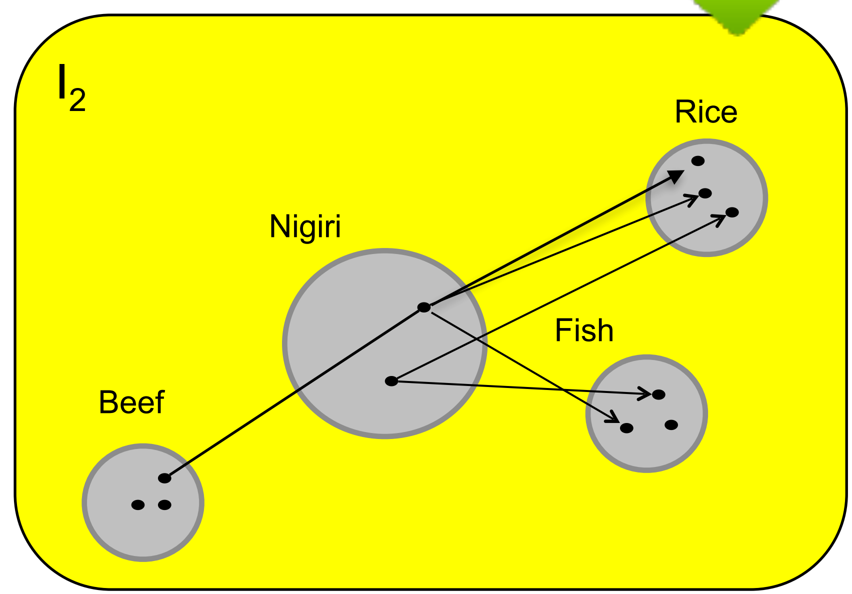
- Does Nigiri contain rice?
- Does Nigiri contain fish?
- Does Nigiri contain beef?

1st Logical Pattern: the **Property Closure Pattern**

Class: Nigiri
SubClassOf Sushi,
hasIngredient **some** Rice,
hasIngredient **some** Fish

Which of these interpretations is a model of the above axiom?



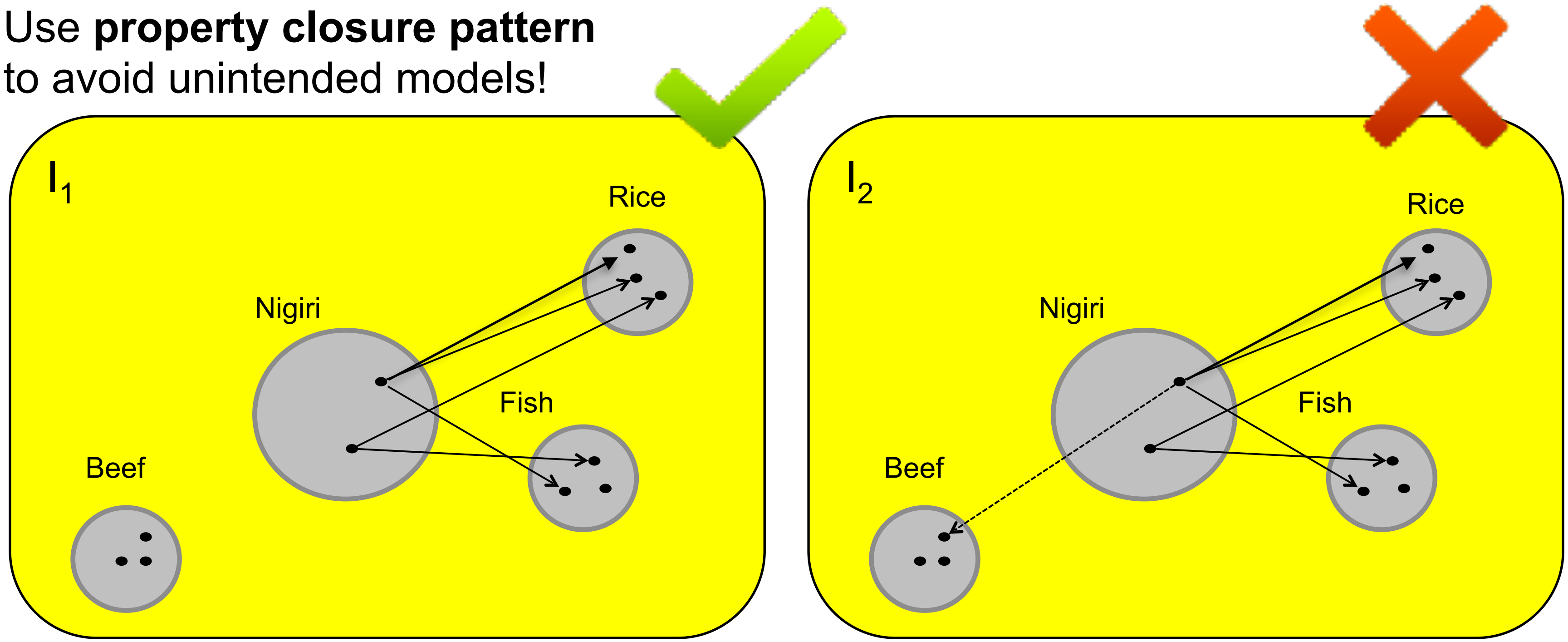


→ hasIngredient

1st Logical Pattern: the **Property Closure Pattern**

Class: Nigiri
SubClassOf Sushi,
hasIngredient **some** Rice,
hasIngredient **some** Fish,
hasIngredient **only** (Fish or Rice)

Use **property closure pattern**
to avoid unintended models!



→ hasIngredient

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

Class: Nigiri
SubClassOf Sushi,
hasIngredient **some** Rice,
hasIngredient **some** Fish

$\mathcal{O} \models$ Nigiri SubClassOf hasIngredient some Rice?

Q: "Does Nigiri have beef as ingredient?"
A: "Maybe/Don't know"

DisjointClasses: Rice, Fish, Beef
Class: Nigiri
SubClassOf Sushi,
hasIngredient **some** Rice,
hasIngredient **some** Fish,
hasIngredient **only** (Fish or Rice)

$\mathcal{O} \models$ Nigiri SubClassOf hasIngredient some Rice?

Q: "Does Nigiri have beef as ingredient?"
A: "No"

1st Logical Pattern: the **Property Closure Pattern**

- In general, 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)
```

MANCHESTER 1824

The University of Manchester

2nd Logical Pattern: the **Covering Pattern**

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

Class: Y1 **SubClassOf** X
Class: Y2 **SubClassOf** X
…
Class: Yk **SubClassOf** X

- 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 a UG, MSc, MRes, or PhD student
- To ensure this **coverage of** X by Y1,…Yk, we use the **covering axiom:**

Class: Y1 **SubClassOf** X
Class: Y2 **SubClassOf** X
…
Class: Yk **SubClassOf** X

Class: X **SubClassOf**: (Y1 or … or Yk)

- **Quick exercise**: translate the above axioms into FOL!

MANCHESTER 1824

The University of Manchester

3rd Logical Pattern: the **Partitions Pattern**

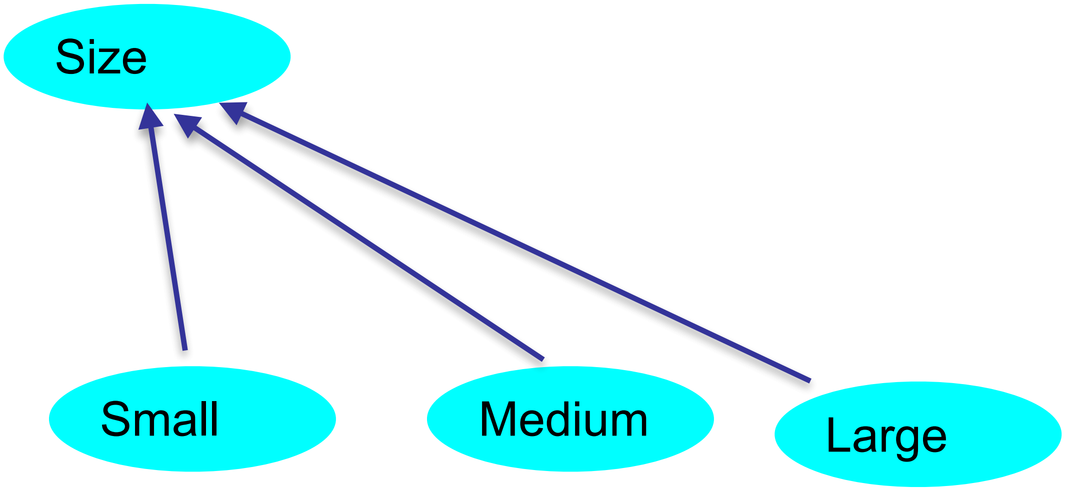
- Say we have Class X with subclasses Yi
  - e.g., UG, MSc, MRes, PhD are all subclasses of Student
- Now we *may* want to say that "no individual can be an instance 2 or more of these class Yi"
- How do we "partition" values **for properties** such as Size, Spicyness, etc:
- E.g., we want to say that 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

3rd Logical Pattern: the **Partitions 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)

Disjoint
+ Covering

Partition

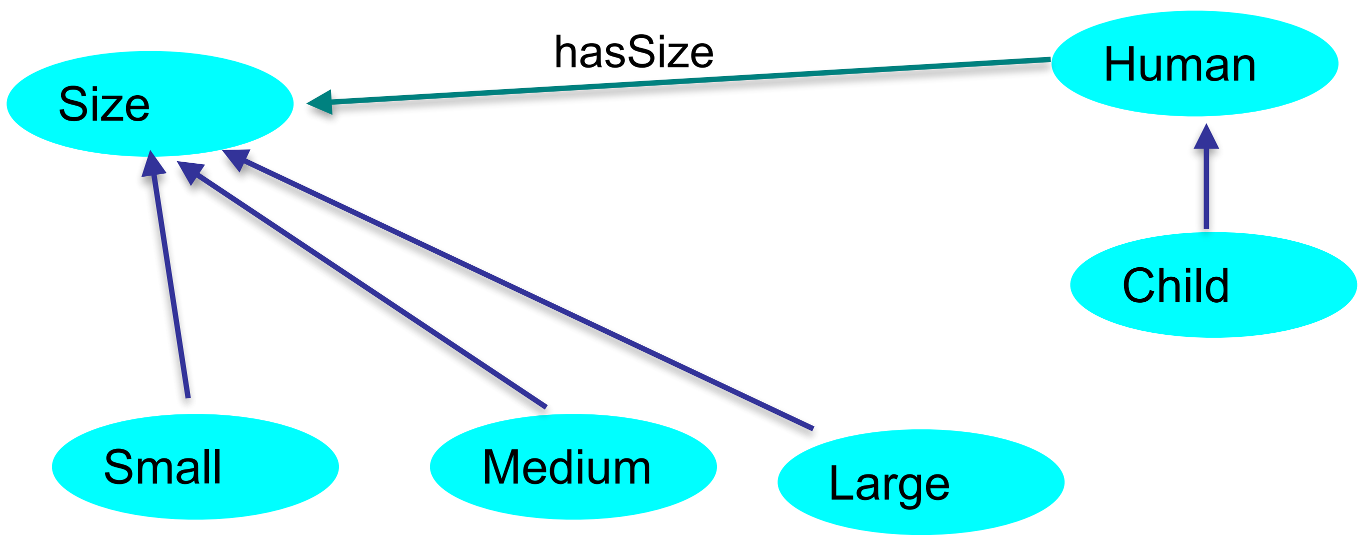


4th Logical Pattern: the **Entity Property Quality Pattern**

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

Partition Pattern



4th Logical 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., hasSize 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
      - may apply via the **domain** of the property and
      - 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

More information on logical patterns….

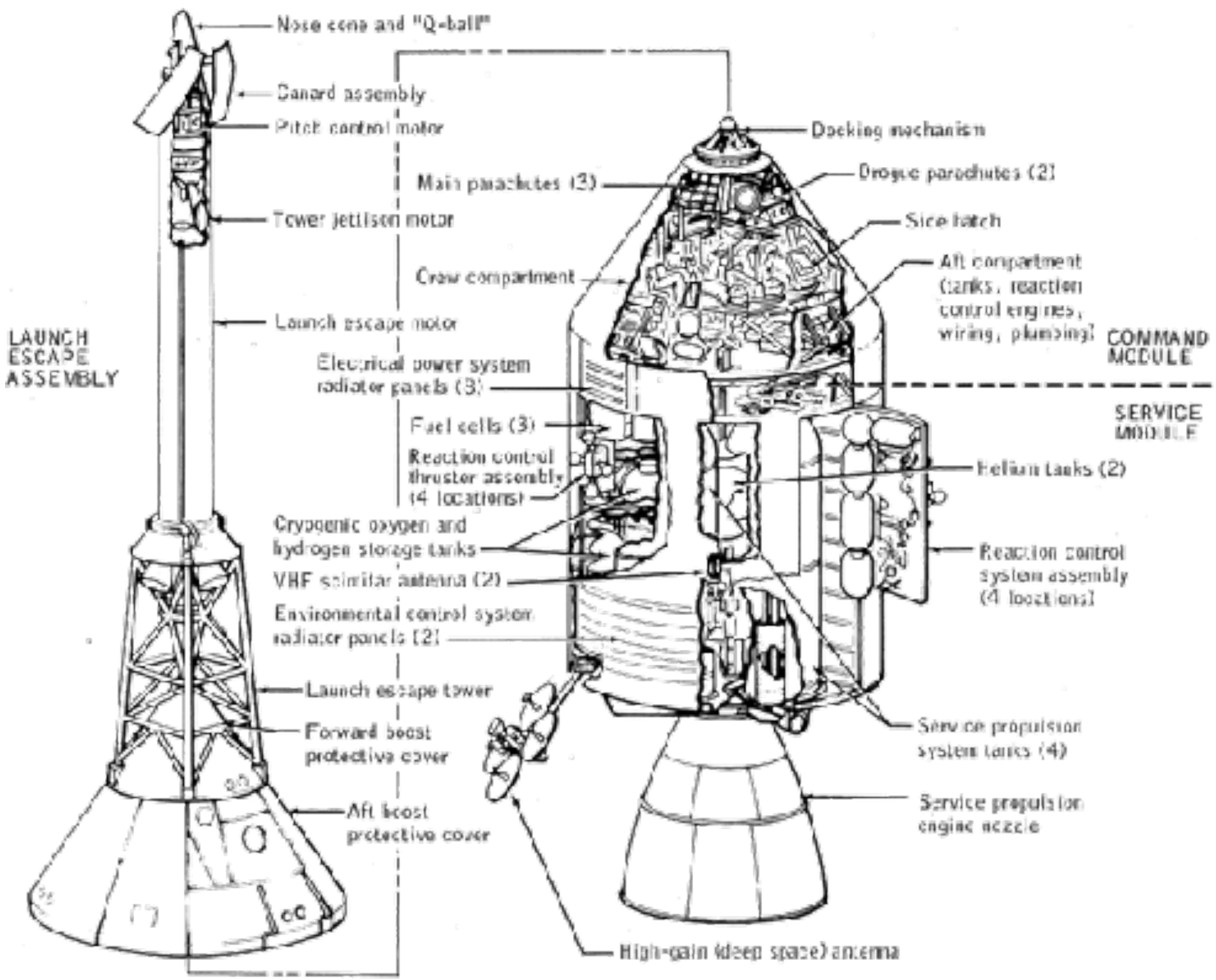
- Have a look at
  - http://www.w3.org/TR/swbp-specified-values/
  - http://ontogenesis.knowledgeblog.org/1499
  - http://ontogenesis.knowledgeblog.org/1001
  - Lots of short, accessible articles about ontology stuff

MANCHESTER
1824

The University of Manchester

Partonomies:
Parts and Wholes

Towards Content Patterns: Composition, Parts and Wholes



MANCHESTER 1824

The University of Manchester

Composition or Aggregation

- Describing a **whole** by means of its **parts,** e.g.,

  AppleCake is a Cake that has parts that are Apple

- Is *hasPart* one or more relations?
  - If more, what are the primary composition relationships?
- What inferences can we make?
- What might we have in our representation languages to support this?

http://www.flickr.com/photos/hartini/2429653007

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

Parts & wholes: examples

Toothbrush — Bristles

Shopping Trolley — Wheels

Car — Iron

Cappuccino — Milk

Kilometer — Meter

England — Manchester

Forest — Tree

Pie — Slide of Pie

Book — Chapter

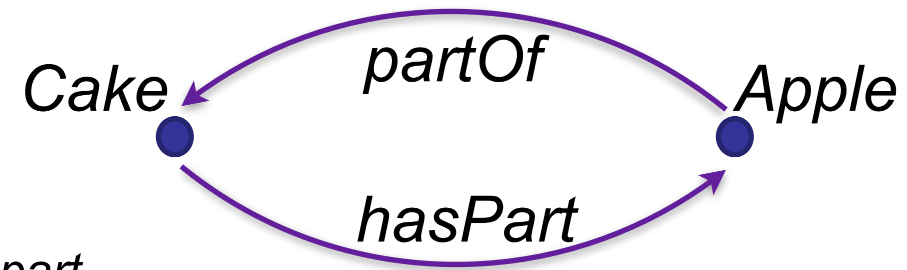
University of Manchester — You

http://www.flickr.com/photos/aramisfirefly/4585596077

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

Is part of versus *has part*

- Of course *is part of* is a **different** relation than *has part*
  - my hand *is part of* me but
  - ~~my hand *has part* me~~



- But *is part of* is the **inverse of** *has part*
  - Protégé makes it easy to say this
  - Not declaring this may cause loss of entailments/inferences
- Semantics: If P is the inverse of Q in $\mathcal{O}$, then for any model $\mathcal{I}$ of $\mathcal{O}$, any x, y in Δ: $(x, y) \in P^{\Delta}$ iff $(y, x) \in Q^{\Delta}$

More on Inverse Properties

- Be careful about what you can/cannot infer around inverse relationships:
- …for example:

Property: hasPart
  InverseOf: isPartOf
Class: Car
  SubClassOf: Vehicle and
    (hasPart **some** Engine)
    (hasPart **exactly 4** Wheel)
Class: Broken
  SubClassOf: Device **and** (isPartOf **only** Broken)

  - does this ontology entail that

Engine **SubClassOf** (isPartOf **some** Car)?
Car **and** (hasPart **some** Broken) **SubClassOf** Broken?

Possible Properties of Part-Whole Relations

- See [Winston, Chaffin, Herrmann1987] and [Odell 1998]
- **functional**:
  - Does the part bear a functional or structural relationship to the whole? Are they in specific temporal/special position to support this functionality?
  - e.g., engine-car, wheel-bicycle
  - Odell calls this “configurational”
- **homeomerous** (homeomeric):
  - Is the part the same *kind of thing* as the whole?
  - e.g., the North-West of England, a slice of bread
- **invariant** (separable)
  - Can the part be separated from the whole (without destroying it)?
  - e.g., a hair of me, the bell of my bicycle
  - often difficult since it involves *identity*
  - e.g. if you remove my arm, I am still me?

1. P-W-R: isComponentOf

- holds between
  - a component and
  - an integral object
  - i.e., a configuration of parts and a whole
- used for a particular arrangement (not just haphazard)

functional
non-homeomeric
separable

- Bristles - toothbrush
- Scene - film
- Handle - CarDoor

- Functional: ripping handle off car door affects functionality (of both)
- Non-homeomeric: handle & door are different kinds of things
- Separable: ripping handle off car door is possible

2. P-W-R: isIngredientOf

- holds between
  - material and
  - object that’s made of this material

non-functional
non-homeomeric
non-separable

- Milk - Capuccino
- Flour - Bread

- Functional: milk is “anywhere” in the cappuccino
- Non-homeomeric: cappuccino and milk are different kinds of things
- Non-separable: can’t take milk out of cappucino/flour out of bread

3. P-W-R: isPortionOf

- holds between
  - a portion and
  - an object

non-functional
homeomeric
separable

- Almost like Material-Object, but parts are *the same kinds of thing* as whole
- aka Slice, helping, segment, lump, drop etc.

- SliceOfBread - Bread
- SomeChocolate - Chocolate

- Non-functional: slices can be anywhere, and don't affect function of whole
- Homeomeric: slide & bread are both bread
- Separable: can cut a slice of bread

4. P-W-R: isSpatialPartOf

- holds between
  - a place and
  - its surrounding area

non-functional
homeomeric
non-separable

- Like Portion-Object, parts are same kind of things as whole
- Unlike Portion-Object, parts cannot be removed

- Manchester - England
- Peak - a mountain

5. P-W-R: isMemberOf

- holds between
  - a thing and
  - a unit/collection of these things

non-functional
non-homeomeric
separable

- Tree - Forest
- Employee - Union
- Ship - Fleet
- I - University of Manchester

- there’s also a non-separable variant “Member - Partnership”:
- e.g., Stan - StanAndLaurel

Summary of Odell's Compositional Relationships

	Functional	Homeomeric	Separable
Component-Integral isComponentOf	Y	N	Y
Material-Object isIngredientOf	N	N	N
Portion-Object isPortionOf	N	Y	Y
Place-Area	N	Y	N
Member-Bunch	N	N	Y
Member-Partnership	N	N	N

P-W-Rs ≠ *Non Compositional Relationships*

- 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
  - I have a bicycle

…a lot of modelling is about making the right distinctions and thus helping to get the right relationships between individuals

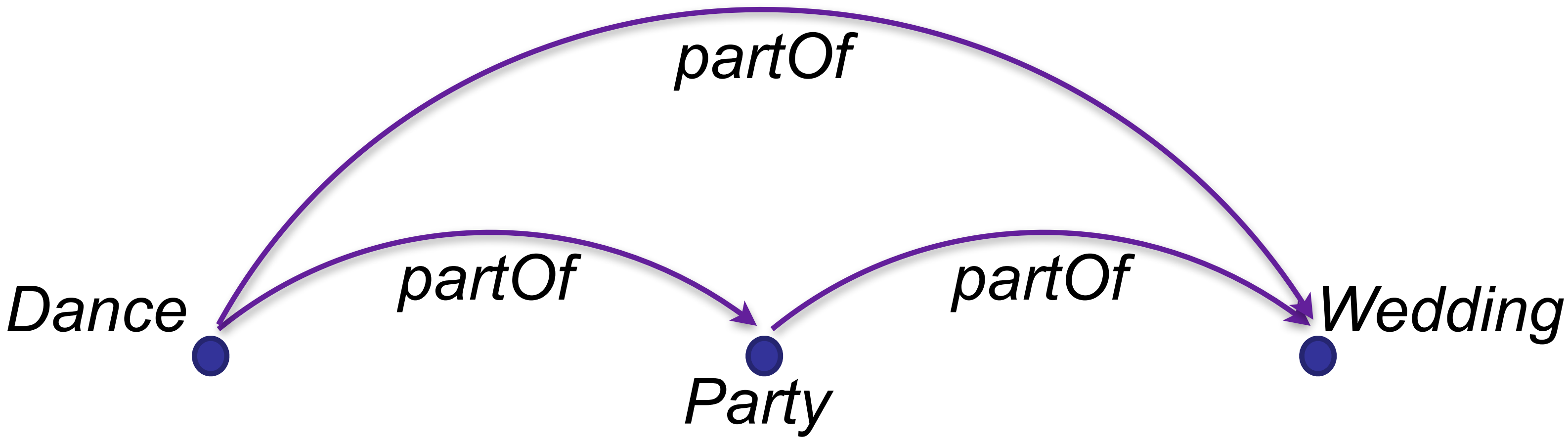
MANCHESTER 1824

The University of Manchester

So what?
Modelling these in OWL

Transitivity

X is part of Y, Y is part of Z,
thus X is part of Z



Transitivity

X is part of Y, Y is part of Z,
thus X is part of Z

- Careful: this is only true for some/with the same kind of composition.

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

- Pistons component of the Engine
- Engine component of the Car
- ➡ Pistons component of the Car

- Sean's arm component of Sean
- Sean member of School of Computer Science
- ➡ Sean's arm component of School of Computer Science
- ➡ Sean's arm member of School of Computer Science
- ➡ Sean's arm part of School of Computer Science

Transitivity

X is part of Y, Y is part of Z,
thus X is part of Z

- Careful: this is only true for some/with the same kind of composition.

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

- Pistons component of the Engine
- Engine component of the Car
- ➡ Pistons component of the Car

Property: isPartOf
Characteristics: Transitive

Property: isComponentOf
SubPropertyOf: isPartOf

Property: isPortionOf
SubPropertyOf: isPartOf
Characteristics: Transitive

- Sean's arm component of Sean
- Sean member of School of Computer Science
- ➡ Sean's arm component of School of Computer Science
- ➡ Sean's arm member 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 **cannot** be transitive
- Solution: provide a transitive superproperty

Property: isPartOf
  Characteristics: Transitive

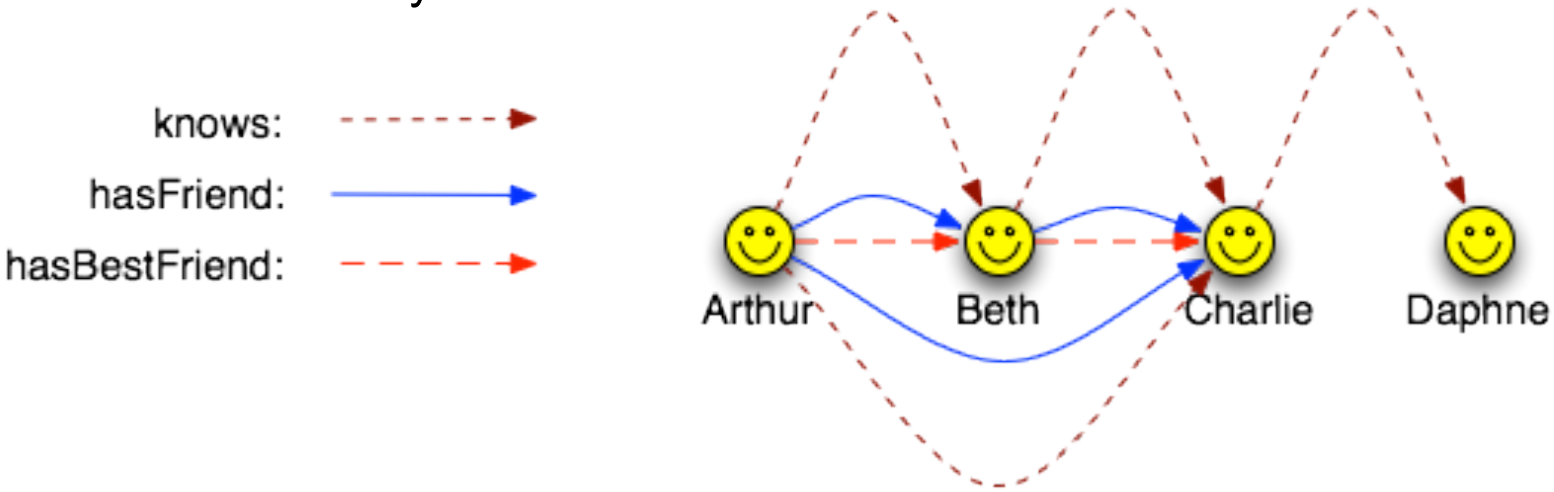
Property: isDirectPartOf
  SubPropertyOf: isPartOf

- 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 Δ:
  - 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

Property: knows
Property: hasFriend
  SubPropertyOf: knows
  Characteristics: Transitive
Property: hasBestFriend
  SubPropertyOf: hasFriend



Generalised Transitivity

- Some P-W relations interact in interesting ways:

- Sean member of School of Computer Science
- School of Computer Science is a portion of the University of Manchester

➡ Sean member of School of the University of Manchester

Property: isPartOf
Characteristics: Transitive

Property: isMemberOf
SubPropertyOf: isPartOf

Property: isPortionOf
SubPropertyOf: isPartOf
Characteristics: Transitive
SubPropertyChain: isMemberOf o isPortionOf

Composition

- Composition provides a mechanism for describing a (class of) object(s) in terms of its parts
- By considering basic properties of part-whole relationships, we can
  - identify different *kinds* of relationship
  - decide where we can (or ca ty.
- Explicitly separating & relating t get correct inferences

Depends on

Property: isPartOf
 Characteristics: Transitive

Property: isLocatedIn
 SubPropertyChain: isLocatedIn o isPartOf
 Characteristics: Transitive

Class Fracture
 SubClassOf isLocatedIn **some** Bone
Class FractureOfFemur
 EquivalentTo Fracture and isLocatedIn **some** Femur
Class HeadOfFemur
 SubClassOf isPartOf **some** Femur

⊨

Fracture and *isLocatedIn* some *HeadOfFemur*

SubClassOf

FractureOfFemur

MANCHESTER 1824

The University of Manchester

Other Content Design Patterns

- …we just talked a lot about how to model composites
- there are many other general content design patterns:
  - how to model time, trajectories, agents, lists, development, roles (see later!), …
- and many domain dependent content design patterns:
  - how to model
    - aquatic resource observations
    - algorithm implementation execution
    - microblog entry
    - hazardous situation
    - …
- See http://ontologydesignpatterns.org/wiki/Main_Page for a long list

Design Principles in OWL:
Multi-Dimensional Modelling
&
Post-Coordination

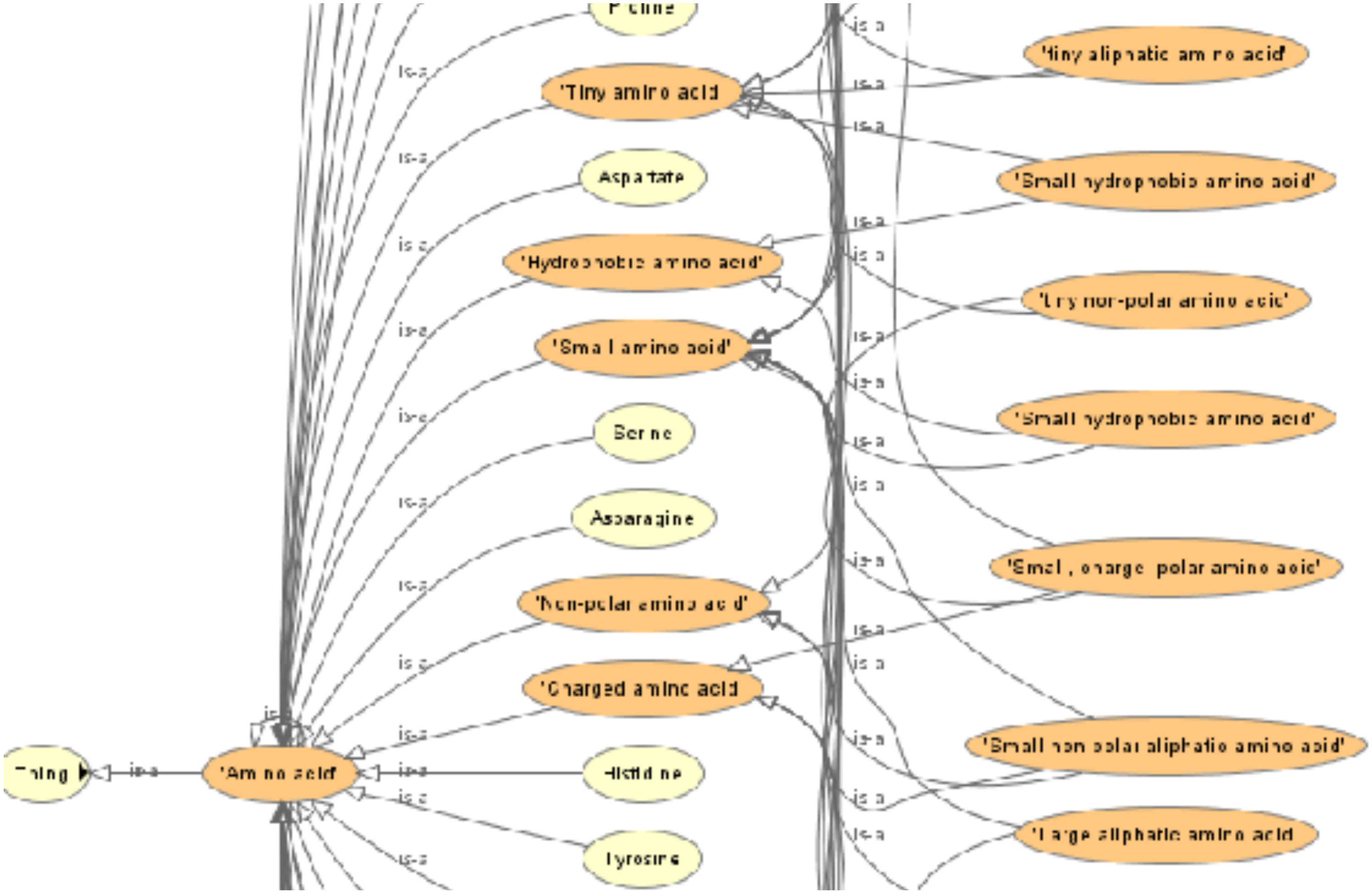
Ontology Normalisation

- 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 medecine

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

Untangeling 1: separate dimensions

Charge
- Negative
- Neutral
- Positive

Polarity
- Polar
- Nonpolar

Size
- Tiny
- Small
- Medium
- Large

Hydrophobicity
- Hydrophobic
- Hydrophilic

Amino Acids
- Alanine
- Arginine
- Asparagine
- Cysteine
- Glutamate
- Glutamine
- Glycine
- Histidine
- Isoleucine
- Leucine
- Lysine
- Methionine
- Phenylalanine
- Proline
- Serine
- Threonine
- Tryptophan
- Tyrosine
- Valine

Untangeling 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

Untangeling 2: relate dimensions using properties

Property: hasSize
Domain: AminoAcid
Range: Size

Property: hasPolarity
Domain: AminoAcid
Range: Polarity

Property: hasCharge
Domain: AminoAcid
Range: Charge

Property: hasHydrophobicity
Domain: AminoAcid
Range: Hydrophilic

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

Polarity
- Polar
- Nonpolar

Size
- Tiny
- Small
- Medium
- Large

Charge
- Negative
- Neutral
- Positive

Hydrophobicity
- Hydrophobic
- Hydrophilic

Untangeling 3: Describe **relevant** terms

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

Untangeling 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

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

Polarity
- Polar
- Nonpolar

Size
- Tiny
- Small
- Medium
- Large

Charge
- Negative
- Neutral
- Positive

Hydrophobicity
- Hydrophobic
- Hydrophilic

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 **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!

Post-Coordination

- Multi-dimensional modelling in OWL allows us to use **post-coordination** and thus avoid tangles like this…

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_ba
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

- if we need all these terms, we can generate them
  - automatically
  - in a principled way
  - ..and update them in case of changes!

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://ontogenesis.knowledgeblog.org/tag/ontology-normalization
    - http://robertdavidstevens.wordpress.com/2010/12/18/an-update-to-the-amino-acids-ontology/
    - http://ontogenesis.knowledgeblog.org/1401

MANCHESTER 1824

The University of Manchester

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

- Thing
  - Process
  - Continuant
    - Dependant_Continuant
      - Property
        - Function
        - Disposition
        - Role
        - Quality
    - Indepentant_Continuant_-_Self-standing
      - Inmaterial
        - Site
        - Information
      - Material

PIMPS: A Simple Domain Neutral Ontology

- Thing
  - **P**rocess
  - **I**mmaterial
  - **M**aterial
  - **P**roperties
    - Quality
    - Function
    - Role
    - Disposition
  - **S**ites

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, breathing, liberation, election
  - Lots of “-ation” and “…ing” words

- **Material**
  - Self-standing things I can “hold in my hand”
  - E.g., ball, car, person, leg, pizza, piece of seaweed
  - 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

MANCHESTER 1824

The University of Manchester

PIMPS: A Simple Domain Neutral Ontology

- **Immaterial**
  - Self-standing things I can **not** “hold in my hand”
  - E.g., idea, goal, 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., HeatResistence
- **Site**
  - point or area on/of a material entity
  - e.g., the area occupied by Manchester
  - 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

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*