Formalizing Subsumption Relations in Ontologies using Type Classes in Coq

Richard Dapoigny[1] Patrick Barlatier[1]

[1]LISTIC/Polytech'Savoie
University of Savoie, Annecy, (FRANCE)

Contents

Definition

- In Computer Science, an ontology formally represents knowledge as a collection of concepts within a domain together with the relationships between pairs of concepts.
- Ontologies are now under development and/or in use in diverse areas such as geography, astronomy, defense, the automotive and aerospace industries, and the life sciences, where they are used to formalize biological and medical terminologies.

Ontologies and Reasoning

- Ontologies can be used to model a domain and support reasoning about concepts and relations.
- Various logic-based formalisms and automated reasoning techniques are being used in order to facilitate such understanding and in order to help the domain experts to construct, maintain, and use ontologies.

In the area of logic-based formalisms, this work is an investigation of the interplay between conceptual analysis for ontologies and their related reasoning problems.

Core ontologies vs Domain ontologies

Ontologies are (at least) divided in two abstraction levels, i.e., core (Foundational) ontologies and Domain ontologies.

- Core ontologies (e.g., SUMO, DOLCE, BFO) contain the basic knowledge specifying what are kinds, relations, etc. (similar to set theory for classical mathematics)
- Domain ontologies (e.g., SNOMED-RT) focus on a particular area such as Biology, Law, etc. and include a single core ontology.

What is Subsumption?

Assumption: two formal relations (*is_a*, part-whole).
Using both *is_a* hierarchies (taxonomies) and part-whole hierarchies (partonomies) requires a clear understanding of these relations.

The Object-Oriented view:

- If a class A is subclass of class B:
  - Every instance of A is also an instance of B
  - Values of properties of B are inherited by instances of A
- There are many examples where the use of subclass-of relation can be incorrect in subtle ways.

Another way is to separate properties from classes e.g., in DL $\Rightarrow$ we share this view.

What is Subsumption?

- In DL, subsumption is restricted to *is_a* relations.
- The conceptualization of *is_a* and part-whole relations is inconsistent and problematic while in some cases, they are not clearly distinguished [Smith04].
- It could be appealing to conceive them as two forms of subsumption [Rast04].
- We defend the position here, that (i) these relations stem from a common general relation and (ii) that all relations can be expressed in terms of part-whole relations.

Language Requirements

1. We are seeking for a representation language able to model general concepts and elements of a domain (i.e., an ontology) with different accuracy levels (e.g., the subsumption relations) such that the following properties hold:
   - Communication and knowledge sharing -> provide a common vocabulary
   - Knowledge reuse -> Common knowledge (e.g. time and spatial concepts) which can be reused when building a domain specific applications.

Language Requirements

2. This language should be able:
   - to check for conceptual errors during design (well-formed terms),
   - to infere concepts or relations from an existing model, and
   - to build requests over the ontology.

-> Logic Inferencing/reasoning -> deduce implicit knowledge from explicit knowledge.

So far, current answers use ontologies and involve multiple languages which rely on distinct theories (modal logic, first order logic, Description Logics, etc.) not necessarily compatibles between each other which means: waste of time during translation, information lost, and so forth.

Significant Approaches for Ontology Modelling

Ontology :
Knowledge Representation
+ Logic

Object-Oriented Modelling

Classes, Attributes, Methods, Inheritance, Polymorphism

pros
High expressiveness (typing, inheritance, polymorphism)
Tractabitlity

cons
Poor reasoning
Embedded attributes
Problems with part-whole representation

Description Logics

Concepts, Roles, Operators
+ FOL fragment

pros
Inheritance hierarchies (concepts & roles)
Dynamic attributes
Scaling

cons
Limited expressiveness
Limited reasoning (concept satisfiability, subsumption)
Problems with part-whole representation

Significant Approaches for Ontology Modelling

Ontology :
Knowledge Representation
+ Logic

Object-Oriented Modelling

Classes, Attributes, Methods, Inheritance, Polymorphism

pros
High expressiveness (typing, inheritance, polymorphism)
Tractabitlity

cons
Poor reasoning
Embedded attributes
Problems with part-whole representation

Description Logics

Concepts, Roles, Operators
+ FOL fragment

pros
Inheritance hierarchies (concepts & roles)
Dynamic attributes
Scaling

cons
Limited expressiveness
Limited reasoning (concept satisfiability, subsumption)
Problems with part-whole representation

KDTL

The Knowledge-based Dependently Typed Language

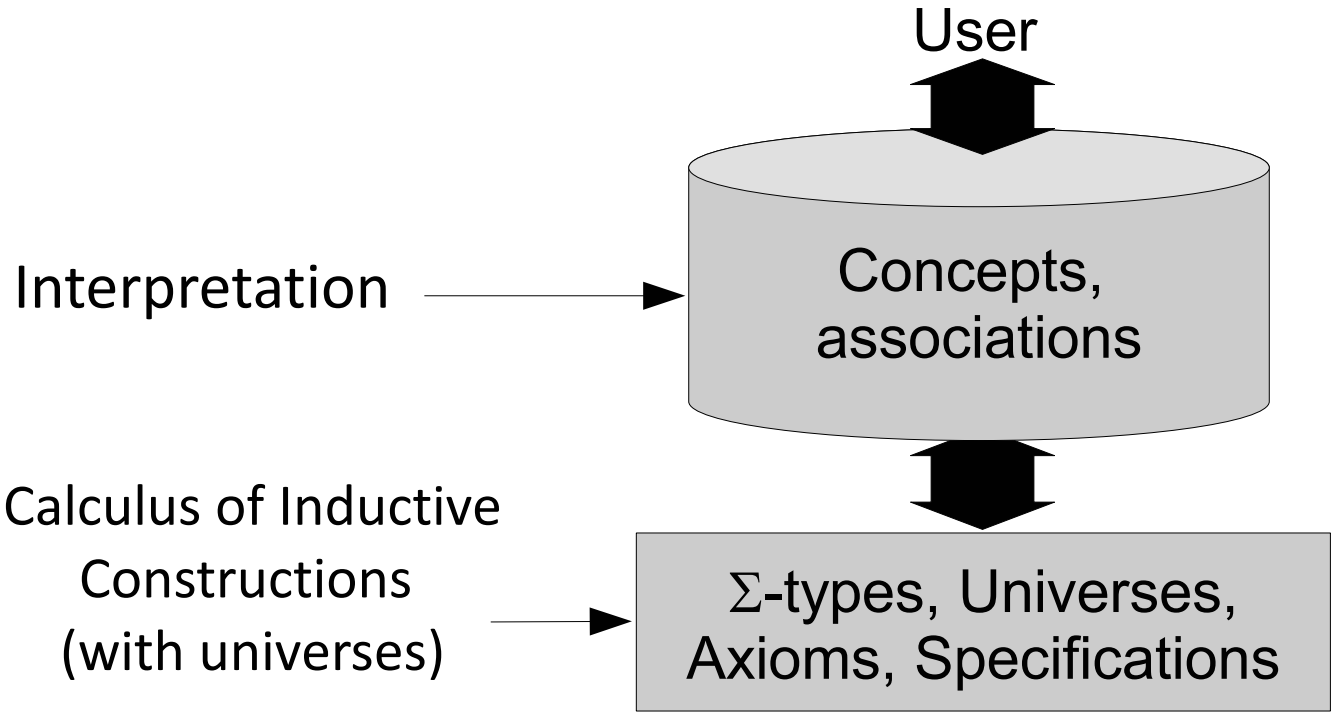
Why using a Dependent Type Theory?

We suggest a language (KDTL) rooted in dependent-type theory.

1. Providing high expressiveness and to enforce semantic conditions.
2. Universes closed under type-forming operations for distinguishing different parts.
3. Higher-order to permit instances of categorization types to be types themselves and to directly support quantification over sets and general concepts (expressive reasoning).
4. Type inference should help to avoid misconceptions (well-formed concepts).
5. Existence of available tools (e.g., Coq) making more exploitable the underlying theory.

KDTL

Architecture:



Interpretation

Names of ontological categories are interpreted as types using the symbol $[\![\cdot]\!]$ as follows:

i. The context $\Gamma_{\mathcal{O}}$ is such that it includes all terms which are interpretations of universals (e.g., types or universes) in $[\![\mathcal{O}]\!]$,

ii. any particular $p \in \mathcal{P}$ is interpreted as the proof object $\Gamma_{\mathcal{O}} \vdash p : [\![U]\!]$, such that $\forall\, p \ : \ [\![U]\!]\ (\nexists\, p' \mid p' \ : \ [\![p]\!])$ with $[\![U]\!] : \mathit{Type}_0$, the type which interprets the universal U related to p.

A particular

$h1 : \mathit{HumanHeart}$

HumanHeart is well-formed iff $\Gamma_{\mathcal{O}} \vdash \mathit{HumanHeart} : [\![U]\!]$ for some universal U.

Interpretation

iii. any kind K is interpreted as $\Gamma_{\mathcal{O}} \vdash [\![K]\!] : [\![U]\!]$ with $[\![U]\!] : Type_i$ with $i > 0$ and $[\![U]\!]$ which interprets the universal U.

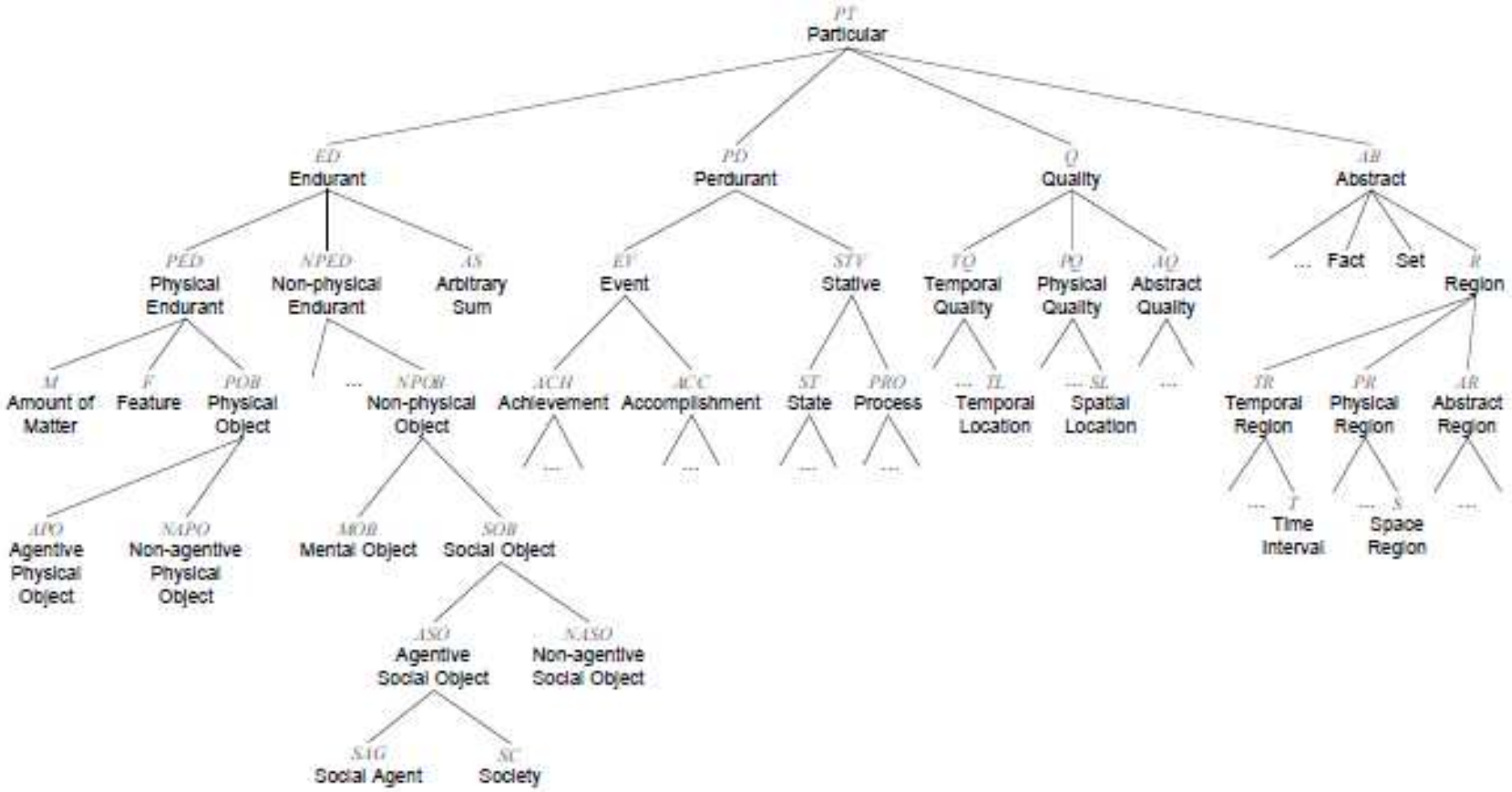
To illustrate the discourse, the hierarchical taxonomy of non-dependent kinds is borrowed from the DOLCE hierarchy [Masolo03] and corresponds roughly to "natural types".

A kind

$HumanHeart : APO$

Interpretation

The DOLCE Backbone



Interpretation

iv. any association[1] σ relating universals U_1 and U_2 is interpreted as the nested Σ-type:
$\sigma : (\Sigma x_1 : [\![U_1]\!] \,.\, \Sigma x_2 : [\![U_2]\!] \,.\, \Sigma x_3 : [\![R]\!][x, y] \;\ldots\; \Sigma x_i : P_i(x_3, \ldots))$,
with $[\![R]\!] : ([\![U_1]\!] \rightarrow [\![U_2]\!] \rightarrow Prop)$.

The PartWhole (binary) association:

$$Relation : Kind \rightarrow Kind \rightarrow Prop$$

$$PartWhole : \Sigma p : Kind . \Sigma w : Kind . \Sigma pw : Relation[p, w] . RefAntisym[pw]$$

[1] Ontological relation.

Interpretation

v. any property P in $[\![\mathcal{O}]\!]$ is interpreted as the kind $[\![P]\!] : Q$ or associations.
   - Properties depend on a concept (kind) to which they apply.
   - The collection of properties *CP_Kind* for a given kind is given by the parameterized nested Σ-type:

$$[\![CP\_Kind]\!][x : [\![K]\!]] := (\Sigma x_1 : [\![K_1]\!]. \ldots, \Sigma x_i : [\![K_i]\!] \,.\, \sigma[x, x_i], \ldots)$$

A collection of properties

$$CP\_HumHeart[x : HumanHeart] := (x_1 : Volume \times x_2 : Age \times \Sigma x_3 : Pump \,.\, FunctionAs[x, x_3])$$

Formal Associations

Since properties are separated from universals:

- → Properties can be shared by several universals
- → Properties inhere in a given concept instance
- → Properties are dynamic w.r.t. the instance
- → Hierarchies of conceptual structures can be based both on properties and part-whole associations

Formal Associations

Definition

Given two universals U and V together with their respective collections of properties $Pr(U)$ and $Pr(V)$, if U *is_a* V holds, then the association is persistent that is, $Pr(V) \subseteq Pr(U)$.

Definition

Given two universals U and V together with their respective collections of properties $Pr(U)$ and $Pr(V)$, if U is a part of V then the association is non-persistent that is, $Pr(V) \nsubseteq Pr(U)$.

Using the Coq Language

- Its underlying theory is coherent with the expected profile (constructivism, higher-order, dependent types, universe hierarchy).
- It is one of the two most successful systems in use today [Wiedijk07].
- Coq has a large active user base.
- The Coq language has reached a state where it is well usable as a research tool [Bertot04].
- Very powerful primitives: Types Classes (TCs) [Sozeau08,Spitters11] for describing data structures and inheritance.

The hierarchy of Universals

- The hierarchy of kinds does not make use of properties since they are general concepts and the description of their properties is not relevant.
- The major consequence is that persistence is left implicit here.
- The domain ontology is then built by adding domain-based kinds to the hierarchy.

The hierarchy of Universals

- We define a universe *Universal* and a sub-universe for *Kinds*:

```
Definition Universal                    := Type.
Definition Kind          :  Universal := Type.
```

- Universes are available in Coq, but hardly manipulable.
- To overcome this difficulty, two parametric universes are created for a foundational and a domain ontology:

```
Definition OntoCore (c:Type)            := Type.
Definition OntoDomain (c:Type)          := Type.
```

The hierarchy of Universals

- Then each DOLCE kind is typed with its parametric universe using coerced structures (the syntax `:>` in the definition of structures does not imply backward reasoning as in type classes).
- The above fragment details this mechanism on some DOLCE categories showing e.g., that every Perdurant (PD) and every Endurant (ED) are also particulars (PT).

```
Structure PT   :  OntoCore Kind    := {}.
Structure PD   :  OntoCore PT      := {PDsub :> PT}.
Structure ED   :  OntoCore PT      := {EDsub :> PT}.
Structure AB   :  OntoCore PT      := {ABsub :> PT}.
Structure R    :  OntoCore AB      := {Rsub :> AB}.
 ...
```

PartOf Associations (i.e., transitive part-whole relations)

The Part-of Association between Universals:

```
Parameter Relation :  (OntoCore Kind) → (OntoCore Kind) → Prop.

Class PartOf {X Y :OntoCore Kind}        :  Prop    := {
                PartOf_s                 :> Relation X Y;
                PartOf_pred              :> POR}.
```

The Part-of Association between Individuals:

```
Parameter IndRelation :  PT → PT → Prop.

Class IndPartOf {X Y : PT}               :  Prop    := {
                pPartOf_s                :> IndRelation X Y;
                pPartOf_pred             :> POR}.

Axiom SubRel :  forall (X Y:OntoCore Kind)(x y:PT),
                          Relation X Y -> IndRelation x y.
```

PartOf Associations (i.e., transitive part-whole relations)

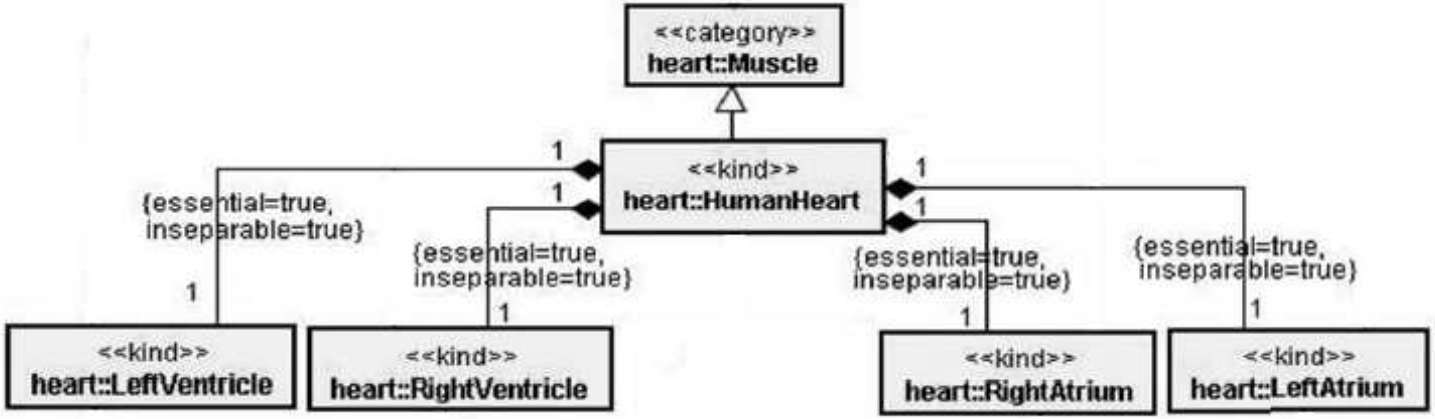
The following lemma states that part-of associations between universals propagates to individuals:

```
Lemma subPartOf :  forall (X Y:OntoCore Kind)(x y:PT)
                        (c:@PartOf X Y), @IndPartOf x y.
  Proof.
  intros.
  decompose record c.
  apply SubRel with X Y x y in PartOf_s0.
  intuition.
  Qed.
```

- Part-whole associations between universals allow for generic statements.
- Part-whole associations for individuals can be restricted to proper part-of.

Type classes depending on the whole provide properties as fields.

```
Class P_HumHeart {x1:LeftVentricle}{x2:RightVentricle}
                 {x3:LeftAtrium}{x4:RightAtrium}
                 (W:HumanHeart) :  Prop := {
                      LVPOHH : @IndPartOf x1 W;
                      RVPOHH : @IndPartOf x2 W;
                      LAPOHH : @IndPartOf x3 W;
                      RAPOHH : @IndPartOf x4 W }.
```



Ontological entities	*Conceptual entities*	*KDTL entities*
concept	Kind	LeftVentricle : OntoDomain PT
collection of properties	Nested Σ (Kind, Association)	Class CP_HumHeart {x1:LeftVentricle} (W:HumanHeart) : Prop := LVPOHH : @IndPartOf x1 W.
association	Association	Class IndPartOf (X Y:PT) : Prop := { pPartOf_s :> IndRelation X Y; pPartOf_pred :> POR}.

Differentiating (ontological) part-whole and *is_a* associations

- The fundamental property of *is_a* association is true of two concepts K_1 and K_2 iff their related sets of field in the corresponding TCs are in the inclusion relation $\rightarrow$ Coq implicitly assumes this property with parameters in TCs or coerced records.
- Part-of associations being described with type classes as a partial order relation, they can be ranked using inheritance between TCs while their meta-properties are inserted as fields in generic part-of relations.
- In addition, basic cardinality constraints can be defined:
  - [1..1] with simple type declaration `x : T`
  - [0..1] with `option` type
  - [0..*n*] with `list T`

Transitivity

```
Class Antisymmetric :  Prop                    :=
        Antisymmetry :  forall (x y :  OntoCore Kind),
                    Relation x y → Relation y x → x = y.
Class Reflexive     :  Prop                    :=
        Reflexivity :  forall x :  OntoCore Kind,
                    Relation x x.
Class Transitive    :  Prop                    :=
        Transitivity :  forall x y z :  OntoCore Kind,
                    Relation x y → Relation y z → Relation x z.

Class POR           :  Prop                    := {
        POR_Refl                               :> Reflexive;
        POR_Antisym                            :> Antisymmetric;
        POR_Trans                              :> Transitive}.
```

On-going work

Part-whole associations can be ranked according to their meta-properties:

- Classification on the basis of the nature or ontological category of related wholes and parts.
- Classification using orthogonal characteristics of the relation (meta-properties):
  - Transitivity: gives rise to part-of associations.
  - Functional: w.r.t. the functional role of parts and whole.
  - Homeomorous: parts are similar to each other and to the whole to which they belong.
  - Shareable: The part may belong to several wholes at the same time.
  - Mandatory: The part cannot be removed from the whole without destroying the whole.
  - Separable: The part can be removed from the whole and may exist independently of the whole.

Expressing meta-properties

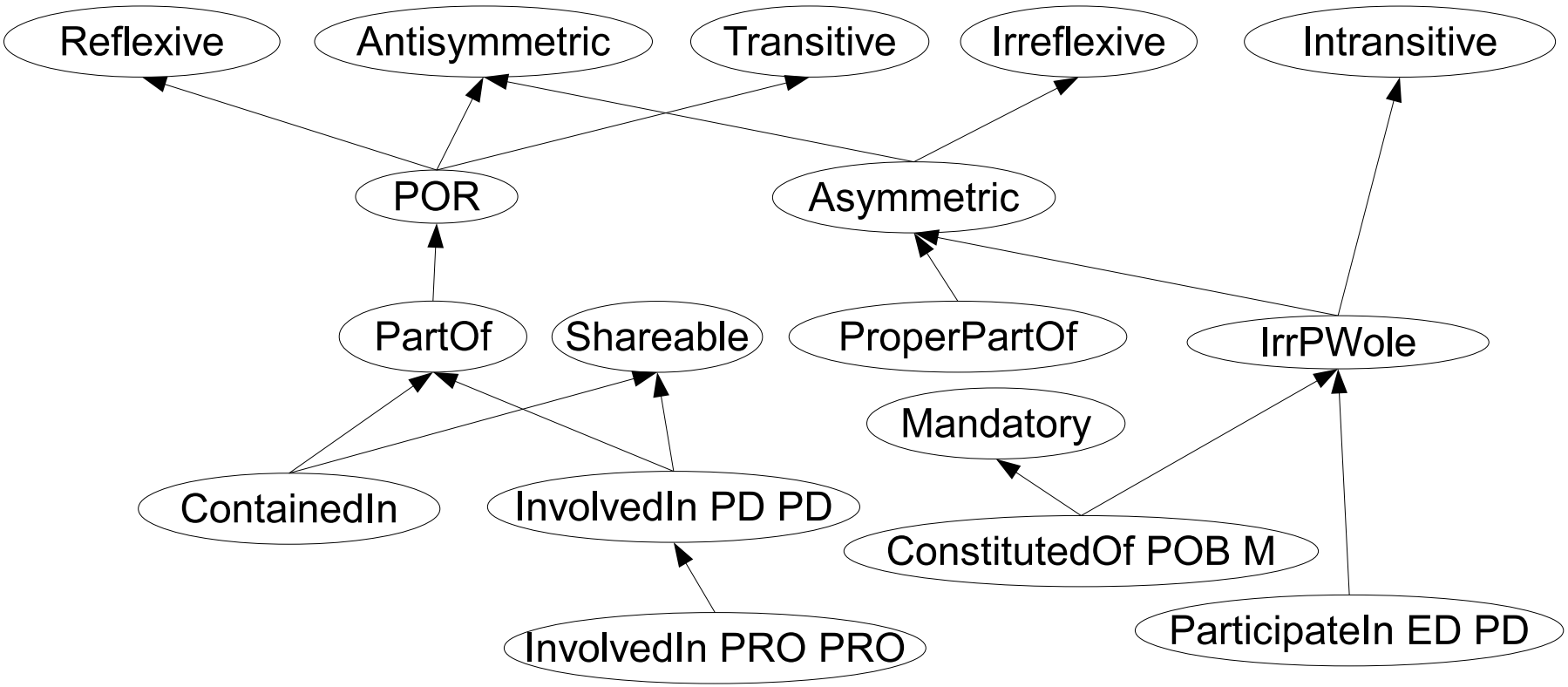
Persistence results from Type Classes inheritance.

Using operational TCs

```
Parameter Function :  PT->Prop.
Class FunctionalPart (P:PT) : Prop:=
                Fp :  forall p w:PT, (Function p ->
                        Function w) -> @IndPartOf p w.
Class ShareablePart (P:PT) : Prop :=
                ShareP : exists W W':PT, @IndPartOf P W ->
                                  @IndPartOf P W'.
```

Expressing meta-properties

We have the ability to build part-whole hierarchies.



Going a Step Further

- Unlike DLs, the number and structure of meta-properties is not limited to a small number of built-in primitives.
- In such a way, meta-properties can be arranged into a hierarchy using type classes.
- Furthermore, tactics can be the building blocks for high-level reasoning.

Incompatible associations

Suppose that we want to prove that an association, e.g., having the type `ProperPartOf LeftVentricle HumanHeart` cannot be a part-of association with swapped arguments.

```
Lemma PPO_not_PO : forall c:@ProperPartOf LeftVentricle
                   HumanHeart, ~(@PartOf HumanHeart LeftVentricle).
Proof.
intros.
decompose record c.
unfold not.
intro p.
decompose record p.
eapply Asymmetry.
exact PPO_s0.
exact PartOf_s0.
Qed.
```

In summary

- The Coq implementation is able to constrain the semantics of knowledge representation based on expressive typed structures.
- The higher order capabilities of the type-theoretical layer are a crucial advantage for meta-reasoning.
- TCs can model several non-trivial aspects of relations such as meta-level properties and multiple inheritance.
- Many aspects of OO programming can be preserved in type theory since it unifies functional programming, component based programming, meta-programming (MDA), and logical verification (see [Setzer07]).
- Much remains to do! i.e., a complete hierarchy of part-whole associations, how meta-properties propagate throughout the hierarchy and ultimately well-formed ontologies.

Thanks for your attention.