Towards Ontological Correctness of Part-whole Relations with Dependent Types

Richard Dapoigny[1] Patrick Barlatier[1]

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

Cite as: Dapoigny, R., Barlatier, P. Towards Ontological Correctness of Part-whole Relations with Dependent Types.

Contents

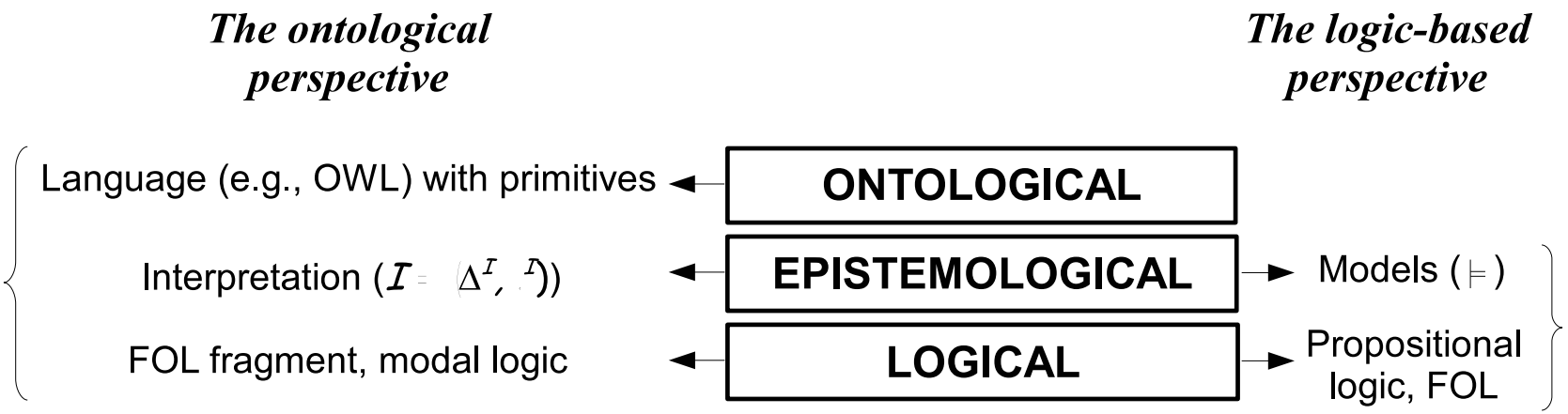
A core problem in ontologies: Part-Whole modeling and reasoning

Some limitations of classical approaches. Usually, reasoning is achieved with Description Logics (DLs) or fragments of First Order Logic (FOL). But there are many challenging problems such as:

- The differences that appear in parthood relations between classes and parthood relations between instances.
- Meta reasoning about Part-whole relations.
- Non-differentiation among the roles that parts play within the structure of a whole.
- Whether the inheritance property holds for parthood relations.

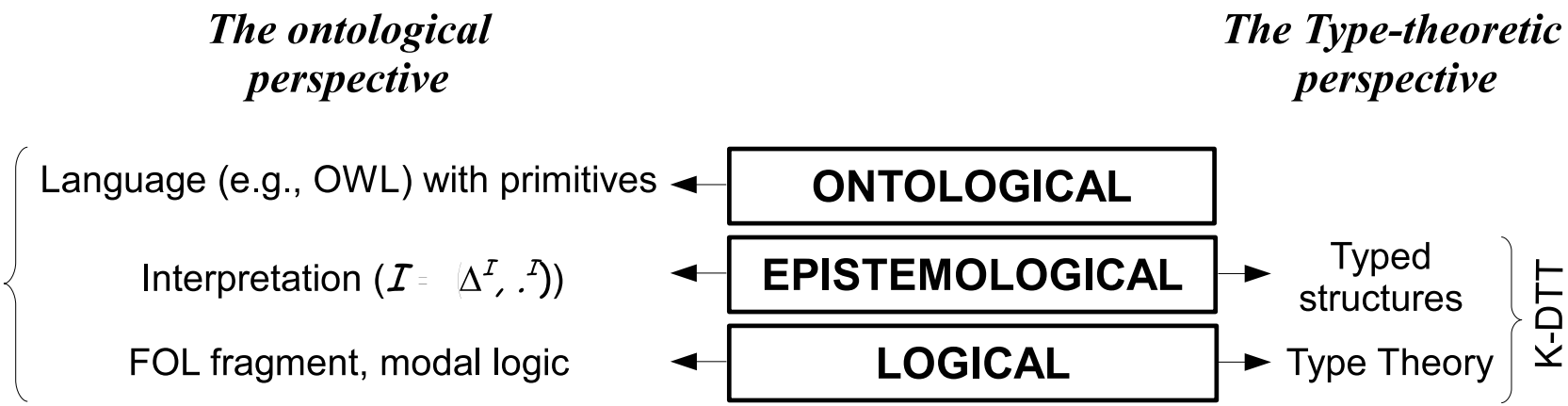
The Layered Model for ontologies

The DL-based model:



The Layered Model for ontologies

The K-DTT model:

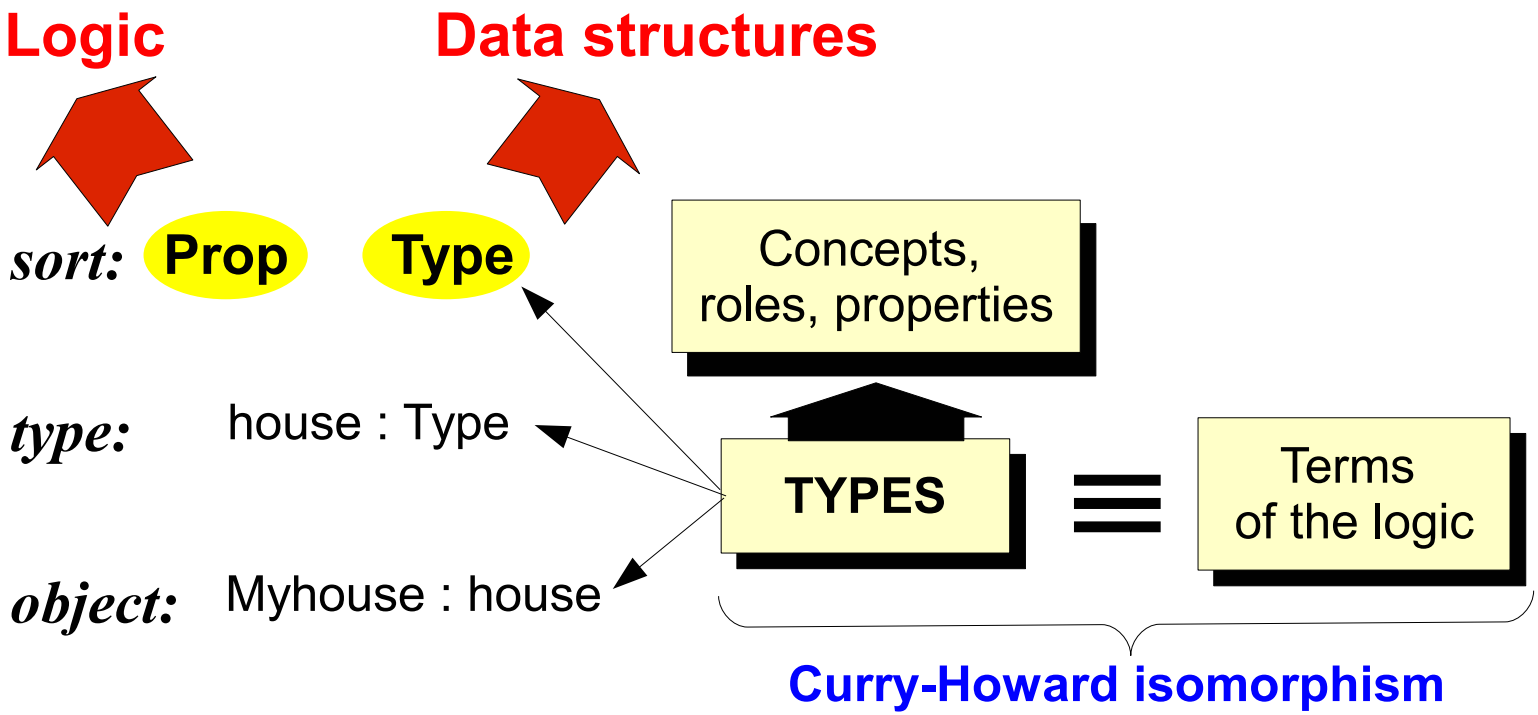


Type Theory

- Dependent type theory: foundation for constructive mathematics.
- Using constructive logic $\Rightarrow$ epistemologically unclear steps in proofs are forbidden.
- In computer science, dependent types $\Rightarrow$ proof assistants and automated theorem provers.
- Dependent type systems mix types and expressions to produce code that is proven to be correct with respect to its expected behavior.
- The Curry-Howard isomorphism: proving $\equiv$ computing.

So, why bother? idea: apply type theory to express and reason about ontological components.

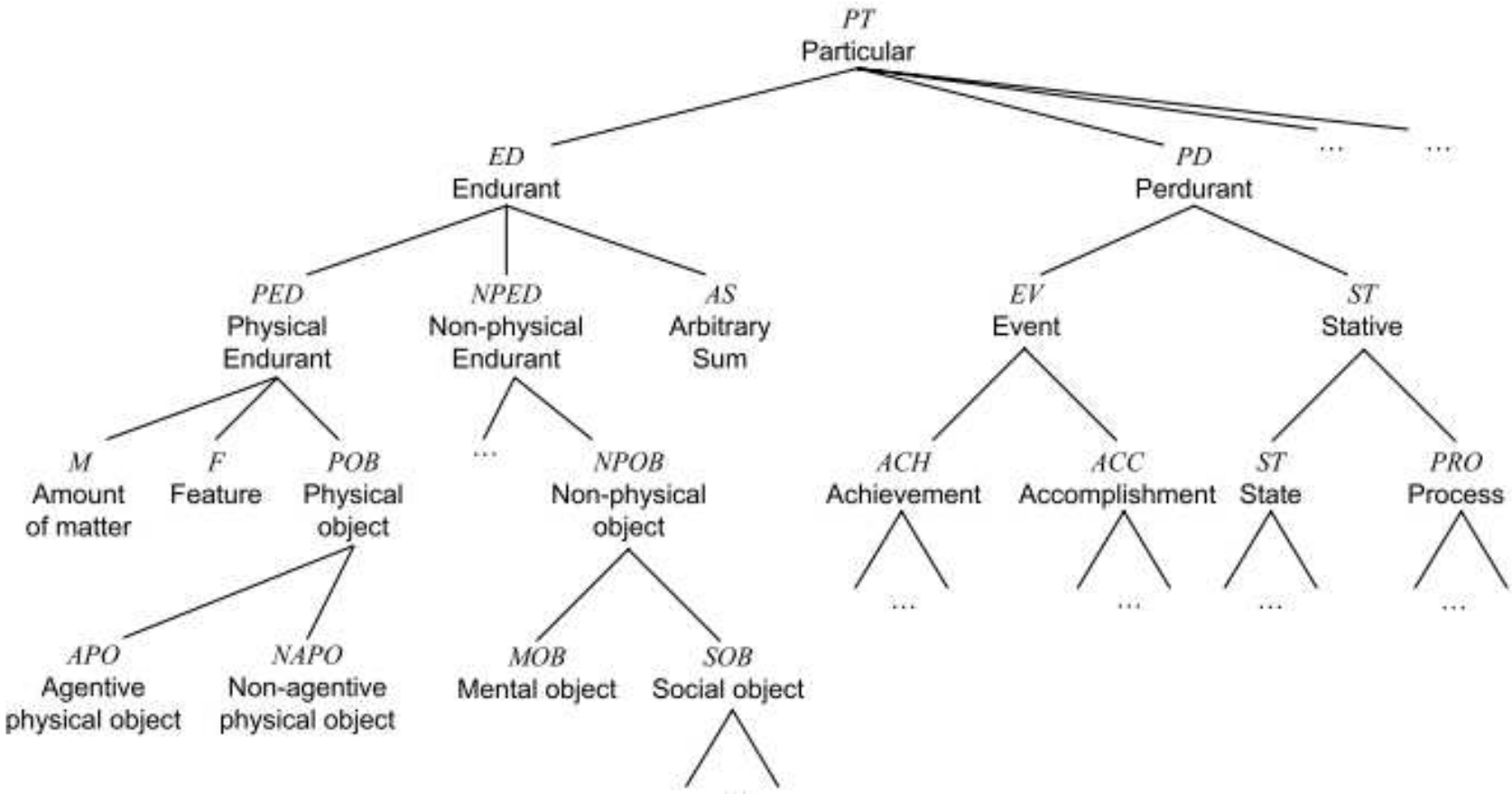
Basis of Type Theory



Universes are stratified (Russell): $Prop \subseteq Type_0 \subseteq Type_1 \ldots \subseteq Type_{i-1}$

Types and ontologies

Stratified universes: $Type_0$ is the universe of "instanciable" types (e.g., *House* is instanciable in *MyHouse*), $Type_1$ the universe for types of instanciable types (e.g., items of the Dolce ontology), and so forth.



Representing knowledge and reasoning with Dependent Types

- Expressing "for all": product types

$$\Pi x : Cell\ .\Pi y : Membrane\ .\ LimitedBy(x, y)$$

- Expressing "for some": sum types

$$\Sigma x : Man\ .\Sigma y : Donkey\ .\ Own(x, y)$$

But we can do more: quantification can be dependent.

Every man who owns a donkey beats it

$$\Pi d : (\Sigma x : Man.\Sigma y : Donkey.Own(x, y)).Beat(fst(d), fst(sec(d)))$$

the domain of quantification of the product (Π) is over the predicate $\Sigma x : Man.\Sigma y : Donkey.Owns(x, y)$.

Expressing knowledge

Relations

- Representing relations with n-ary sum-types.

$$\Sigma x_1 : T_1 \,.\, \Sigma x_2 : T_2 \,.\, \ldots\ \Sigma x_n : T_n \,.\, R(x_1, x_2, \ldots, x_n)$$

where T_1, T_2, ... T_n are types.

Example

$\Sigma x : Person \,.\, \Sigma y : Good \,.\, \Sigma z : Shop \,.\, purchaseFrom(x, y, z)$
with an example of proofs:
$\langle John, \langle PartsbySimons, \langle Amazon, p_1 \rangle \rangle \rangle$
with: $p_1 = purchaseFrom(John, PartsbySimons, Amazon)$

Subtyping

Basic subtyping relation:

$$\frac{\Gamma \vdash M \ : \ A \quad \Gamma \vdash A' \ : \ \mathit{Type} \quad A \leq A'}{\Gamma \vdash M \ : \ A'} \ (\text{Sub})$$

Subtyping of relations: corollary [Luo94]

The relation $\leq$ is the smallest partial order over terms with respect to conversion such that:
if $A \leq A'$ and $B \leq B'$, then $\Sigma x : A \ . \ B \leq \Sigma x : A' \ . \ B'$
Here, A and B can be types or sorts (e.g., Type_0, Type_1, ...)

In K-DTT, subtyping will be used to denote intensional subsumption.

Representing (binary) *part − of* relations

- $Rel(T, T', R) \triangleq \Sigma\, T : Type_i\ .\ \Sigma T' : Type_i\ .\ R(T, T')$ with $R : T \rightarrow T' \rightarrow Prop$.
- Formalizing the binary part-whole relations:

$$PW \quad \triangleq \quad Rel(T_1, T_2, PartOf)$$

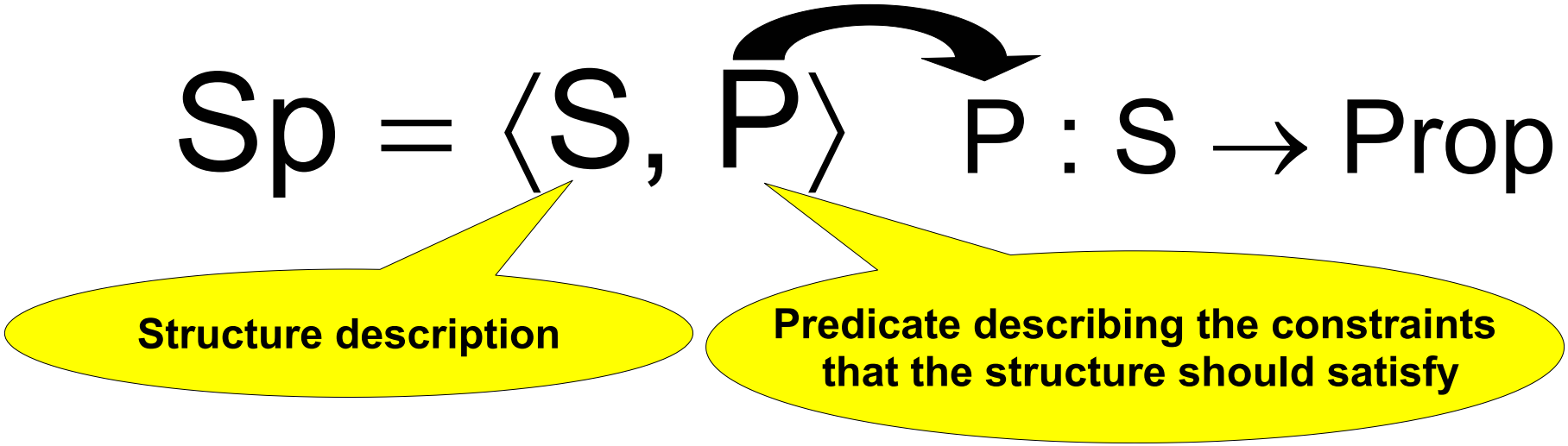
- Example of such a relation:

$$Rel(shaftOfFemur, Femur, PartOf) \triangleq$$
$$\Sigma x : shaftOfFemur\ .\ \Sigma y : Femur\ .\ PartOf(x, y)$$

- How to formally conceptualize the properties of such a relation (e.g., transitivity) within the theory?
- A solution: specifications.

What is a specification?

Specifications have been introduced in theoretical computer science [Burstall-Goguen80, Luo94] for proving the correctness of program modules. A specification *Sp* in K-DTT consists of a pair:



$$Struc[Sp] : Type\ , \quad Pr\ :\ Struc[Sp] \rightarrow Prop$$

Notation

Higher-order logic symbols:

$\top$	$\Longrightarrow$	true (provable in any valid context)
$\bot$	$\Longrightarrow$	absurdity (has no proof)
$\supset$	$\Longrightarrow$	if $P \supset Q$ and P are provable, then so is Q
$\&$	$\Longrightarrow$	if $P \,\&\, Q$ is provable, so are P and Q
$\forall$	$\Longrightarrow$	if $P[x]$ is provable then $\forall x : A.P[x]$ is provable
$\ldots$	$\Longrightarrow$	

Parameterized specification: an example

Provided that a $part - whole$ relation operating on particulars (PT) is transitive, one can introduce parameterized specifications (p-specifications) for transitive relations with:

$$Struc[Trans](Rel(PT, PT, Tr)) \triangleq Rel(PT, PT, Tr)$$

with $Tr : PT \rightarrow PT \rightarrow Prop$

$$Pr[Trans](Rel(PT, PT, Tr)) \triangleq \begin{array}{l} \forall x : PT \,.\, \forall y : PT \,.\, \forall z : PT \,. \\ (Tr(x, y) \;\&\; Tr(y, z)) \supset Tr(x, z) \end{array}$$

Reasoning with transitive relations

If one has a proof (i.e., knows) that a relation is transitive: $Rel(PT, PT, PartOf)$ and since $Valve \leq PT$, $CarEngine \leq PT$ and $Car \leq PT$, then from the corollary, it follows that:

$$\begin{array}{lcl} \Sigma x : Valve \,.\, \Sigma y : CarEngine \,.\, PartOf(x,y) & \leq & Rel(PT, PT, Tr) \\ \Sigma x : CarEngine \,.\, \Sigma y : Car \,.\, PartOf(x,y) & \leq & Rel(PT, PT, Tr) \end{array}$$

Intensional subsumption assumes that these two relations must satisfy the predicative part of their super type, i.e., we get a proof for $PartOf(Valve1, MyCar)$ with $Valve1 : Valve$ and $MyCar : Car$. It means that we have constructed the type:

$$\Sigma x : Valve \,.\, \Sigma y : Car \,.\, PartOf(x,y)$$

Operations on specifications

Fusion of specifications

Given two specifications S and S' having the same structure S_J:

$$\begin{aligned} Struc[Fusion(S, S')] &\triangleq S_J \\ Pr[Fusion(S, S')](z) &\triangleq Pr[S](z) \ \& \ Pr[S'](z) \end{aligned}$$

Extension of specifications

Given a specification $Struc[S]$ and an extent E_S of that structure:

$$\begin{aligned} Struc[Extent(S, E_S, Pr_S)] &\triangleq \Sigma x : Struc[S] \,.\, E_S(x) \\ Pr[Extent(S, E_S, Pr_S)](z) &\triangleq Pr[S](fst(z)) \ \& \ Pr_S(z) \end{aligned}$$

Introducing meta-properties of relations

With the respective p-specifications $Trans(T, T', R)$, $Refl(T, T', R)$, $Antsym(T, T', R)$ that hold for transitive, reflexive and anti-symmetric relations, one can introduce a p-specification POR for partial order relations:

$$
\begin{aligned}
POR(T, T', R) &\triangleq Fusion(Refl(T, T', R), \\
&\quad Trans(T, T', R), Antsym(T, T', R)) \\
Struc[POR(T, T', R)] &\triangleq Rel(T, T', R) \\
Pr[POR(T, T', R)](z) &\triangleq Pr[Refl(T, T', R)](z) \ \& \\
&\quad Pr[Trans(T, T', R)](z) \ \& \\
&\quad Pr[Antsym(T, T', R)](z)
\end{aligned}
$$

with $z : Rel(T, T', R)$.

Introducing meta-properties of relations

Similarly, an intransitive relation *ITR* is such that:

$$
\begin{array}{lcl}
ITR(T,T',R) & \triangleq & Fusion(Asym(T,T',R), ITrans(T,T',R)) \\
Struc[ITR(T,T',R)] & \triangleq & Rel(T,T',R) \\
Pr[ITR(T,T',R)](z) & \triangleq & Pr[Asym(T,T',R)](z)\ \& \\
 & & Pr[ITrans(T,T',R)](z)
\end{array}
$$

and the specification $IRT(PT, PT, PPartOf)$ for proper-part-of p-specifications:

$$
\begin{array}{lcl}
IRT(T,T',R) & \triangleq & Fusion(Asym(T,T',R), Trans(T,T',R)) \\
Struc[IRT(T,T',R)] & \triangleq & Rel(T,T',R) \\
Pr[IRT(T,T',R)](z) & \triangleq & Pr[Asym(T,T',R)](z)\ \& \\
 & & Pr[Trans(T,T',R)](z)
\end{array}
$$

Part-whole p-specifications

Following [Keet08] different types of part-whole relations can be specified,

Generic Part-whole relations

Transitive PartOf: $POR(PT, PT, PartOf) : \quad POR(T, T', R)$
Non-transitive meronymic PartOf:
$NTR(PT, PT, MPartOf) : \quad Asym(T, T', R)$
Intransitive proper PartOf:
$IRT(PT, PT, PPartOf) : \quad IRT(T, T', R)$
Intransitive meronymic PartOf:
$ITR(PT, PT, MPartOf) : \quad ITR(T, T', R)$

Consequence

If a specification S subsumes a specification S', then S' must satisfy all the axioms of S.

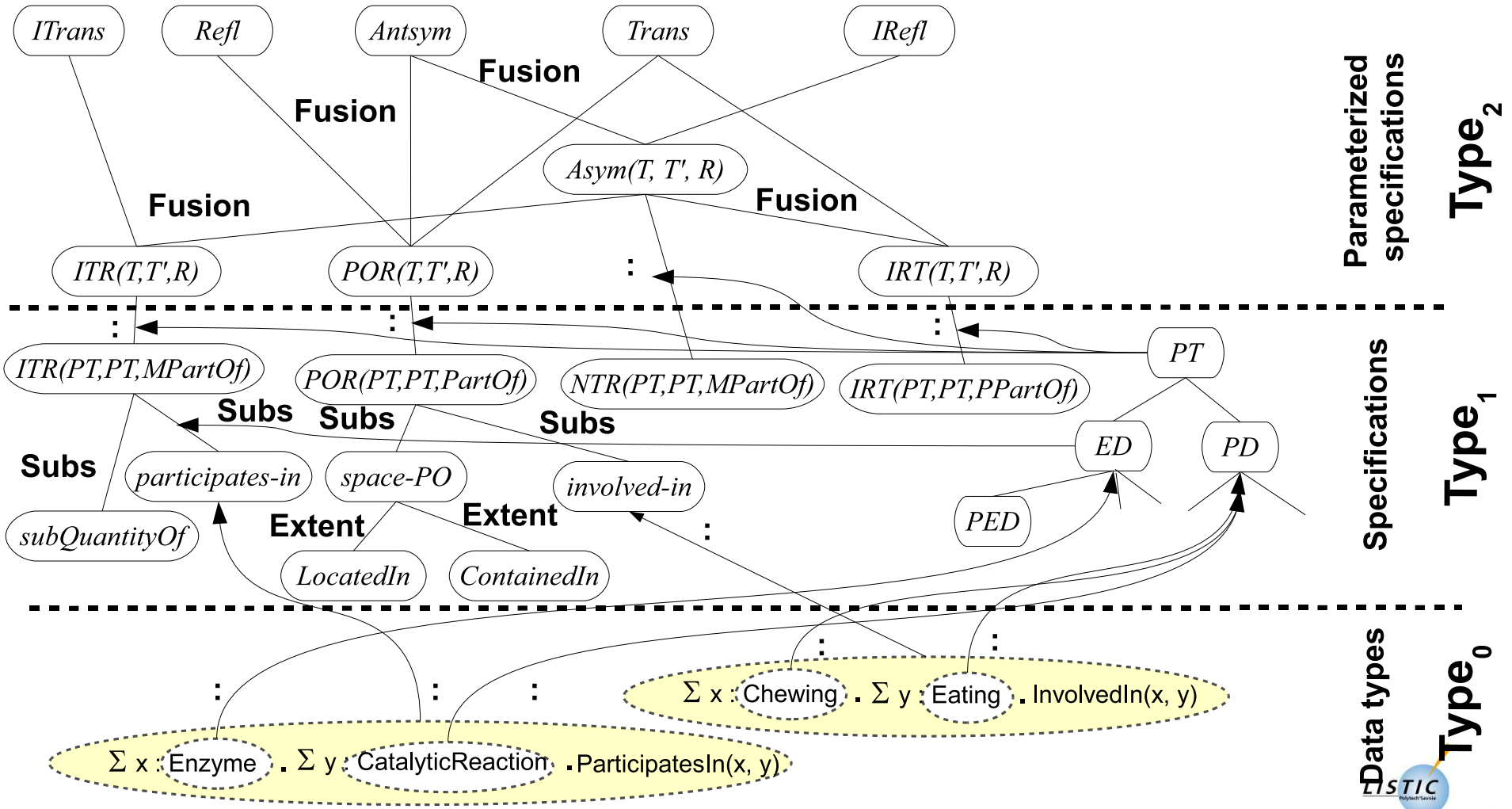
Example

Let us introduce the specification $involved - in$ such that $Struc[involved - in(PD, PD, InvolvedIn)] \leq Struc[POR(PT, PT, PartOf)]$, we have:

$$(\Sigma x : Chewing.\Sigma y : Eating.InvolvedIn(x, y)) : involved - in(PD, PD, InvolvedIn)$$

then $Rel(Chewing, Eating, InvolvedIn)$ satisfies all the predicates of its super-type (i.e., it is reflexive, transitive and antisymetric).

The formal ontology of part-whole p-specifications



Example 1

Correctness of user-defined relations

From DOLCE: $ED \leq PT$ and $PD \leq PT$.
If we add the types *Car* and *CarChassis* such that: $Car : ED$ and $CarChassis : ED$, a part-of relation where *CarChassis* is a part of *Car*.
Since $ED \leq PT$, the relation is of type $POR(PT, PT, PartOf)$, then it is logically correct to speak of part-of relation between *CarChassis* and *Car*.
If $\sigma_1 \triangleq \Sigma x : CarChassis \,.\, \Sigma y : Car \,.\, PartOf(x, y)$, it is possible to automatically derive the inverse relation:

$$\Pi z : \sigma_1 \,.\, HasPart(fst(sec(z)), fst(z))$$

Example 1 (cont.)

Correctness of user-defined relations

If we apply the $involved - in$ relation with the types Car and $CarChassis$:

$$\Sigma x : CarChassis \,.\, \Sigma y : Car \,.\, InvolvedIn(x, y)$$

From the ontology of p-specifications this relation type should be subsumed by:

$$\Sigma x : PD \,.\, \Sigma y : PD \,.\, InvolvedIn(x, y)$$

but $Car \leq PD \supset \perp$ and $CarChassis \leq PD \supset \perp$
then this definition is rejected.

Example 2

Arguments of a relation more general than those of its parent relation

Suppose that we assert:

- The respective argument types are PD and PT for $PartOf$ and $involved - in$.
- $CarChassis : PT$ and $Car : PT$.

The ontology of specifications expects that the following equation holds:

$$\Sigma x : PT \ . \ \Sigma y : PT \ . \ InvolvedIn(x, y) \ \leq \ \Sigma x : PD \ . \ \Sigma y : PD \ . \ PartOf(x, y)$$

The subtyping requires that $PT \leq PD$, which is in conflict with the DOLCE ontology of particulars.

Example 3

High Level Reasoning

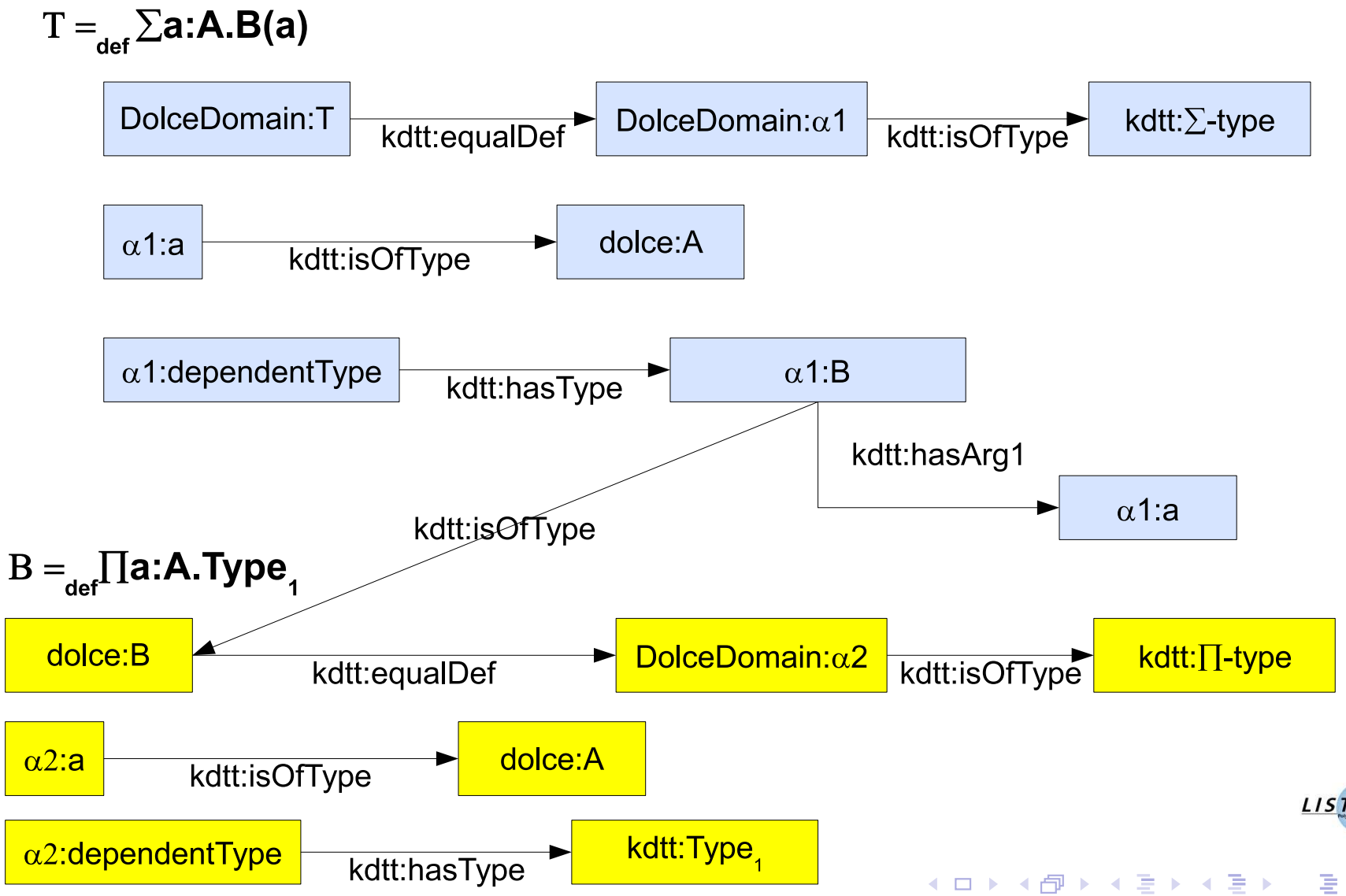
- → Let us define a $subquantity - of$ specification that is intransitive.
- → Its p-specification (in $Type_2$) must be also irreflexive.
- → Choice: $ITR(PT, PT, MPartOf)$ subsuming $subquantity - of$.
- → An instance of the specification $subquantity - of$ is created in the KB.
- → By propagating this instance upwards in the ontology of specifications, an instance of the p-specification $ITrans(T, T', R)$ is created, result: Ok.
- → If a $Type_1$-specification inherits of a specification which generates an incoherence at the $Type_2$ level, then either:
  - a constraint in $Type_2$ is relaxed (e.g., intransitivity)
  - or the $Type_1$-specification is moved across the ontology until it inherits from correct properties.

Ongoing Work

Expressing K-DTT with Triples

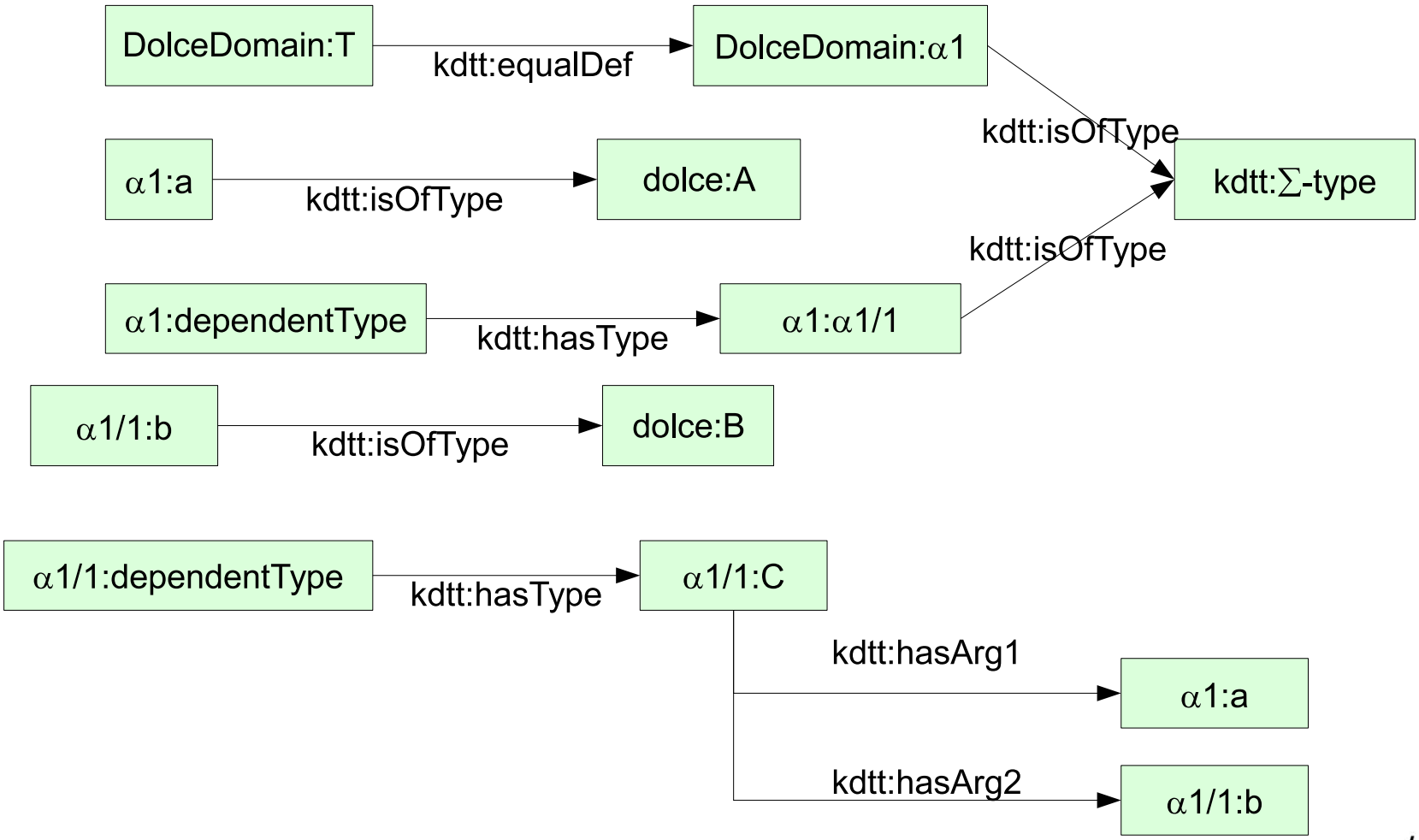
- Any information is expressed in the form of triples store, but without using the semantic related to RDF/OWL since we have designed a dedicated inference engine with AllegroCL.
- The ontology is divided into sub-domains that are mapped into different namespaces with the purpose of distinguishing between the related terminologies.
- The predicate part of triples describe K-DTT operations, e.g., ":" gives *IsOfType*, $\leq$ gives *SubTypeOf* and so forth.
- Data structures are given with the following triples.

Triples


T =def ∑a:A.B(a)
DolceDomain:T
kdtt:equalDef
DolceDomain:α1
kdtt:isOfType
kdtt:∑-type
α1:a
kdtt:isOfType
dolce:A
α1:dependentType
kdtt:hasType
α1:B
kdtt:hasArg1
α1:a
kdtt:isOfType
B =def ∏a:A.Type1
dolce:B
kdtt:equalDef
DolceDomain:α2
kdtt:isOfType
kdtt:∏-type
α2:a
kdtt:isOfType
dolce:A
α2:dependentType
kdtt:hasType
kdtt:Type1

Nested Triples

$T =_{def} \sum a{:}A.(\sum b{:}B.C(a\ b))$



Major Benefits

- Main result: an ontology of p-specifications providing a number of benefits:
  - ⇝ Meta-reasoning is done within a single logic.
  - ⇝ Syntactic differentiation between *partOf* and meronymic relations through p-specifications.
  - ⇝ *partOf* relations are unambiguously identified through their p-specifications with argument restriction.
  - ⇝ The unifying logical theory is able to provide the logical consistency for user-defined structures.
- However, there is a cost of greater difficulty during the modeling stage → a guidance should be designed through, for instance, a question answering process.

Perspectives

- This could be generalized by defining the ontology as a (structured) type and defining rules of inference as inductive relations so as to give a computational understanding of the ontology.
- Specifications can be shared between systems and enhance the expressive power of these systems.

Now, it's time for some questions.