The Distributed Ontology, Model and Specification Language (DOL)
Day 5: Advanced Concepts and Applications

Oliver Kutz[1]
Till Mossakowski[2]

[1]Free University of Bozen-Bolzano, Italy
[2]University of Magdeburg, Germany

Tutorial at ESSLLI 2016, Bozen-Bolzano, August 15 – 19

Summary of Day 4

On Day 4 we have looked at:

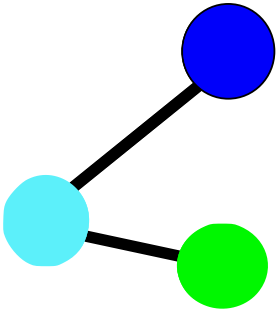
- Semantics of structured OMS
  - based on institutions
- Proofs in OMS
  - based on entailment systems

Today

We will close our introduction to DOL today by introducing several advanced features. These include:

- heterogeneity: working with multiple logical systems
- alignments, expressive bridge ontologies
- networks and combinations of networks
- refinements
- entailment, equivalences, queries

Heterogeneity: Working with Multiple Logical Systems



Example 1: DTV: Can you use these tools together?

The OMG Date-Time Vocabulary (DTV) is a heterogenous* ontology:

- SBVR: very expressive, readable for business users
- UML: graphical representation
- OWL DL: formal semantics, decidable
- Common Logic: formal semantics, very expressive

Benefit: DTV utilizes advantages of different languages

* heterogenous = components are written in different languages

Example 2: Relation between OWL and FOL ontologies

Common practice: annotate OWL ontologies with informal FOL:

- Keet's mereotopological ontology [1],
- Dolce Lite and its relation to full Dolce [2],
- BFO-OWL and its relation to full BFO.

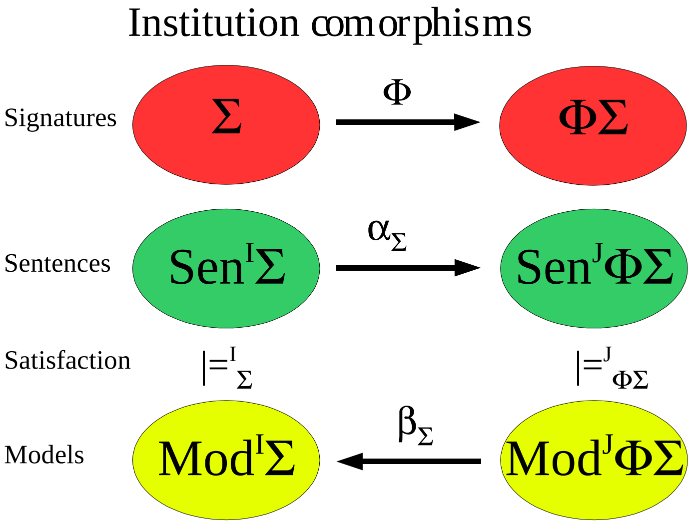
OWL gives better tool support, FOL greater expressiveness.

But: informal FOL axioms are not available for machine processing!

[1] C.M. Keet, F.C. Fernández-Reyes, and A. Morales-González. Representing mereotopological relations in OWL ontologies with ontoparts. In *Proc. of the ESWC'12*, vol. 7295 *LNCS*, 2012.

[2] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Descriptve ontology for linguistic and cognitive engineering. `http://www.loa.istc.cnr.it/DOLCE.html`.

Institution comorphisms (embeddings, encodings)



Institution comorphisms (embeddings, encodings)

Definition

Let $\mathcal{I} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ and $\mathcal{I}' = \langle \mathbf{Sign}', \mathbf{Sen}', \mathbf{Mod}', \langle \models'_{\Sigma'} \rangle_{\Sigma' \in |\mathbf{Sign}'|} \rangle$ be institutions. An institution comorphism $\rho \colon \mathcal{I} \to \mathcal{I}'$ consists of:

- a functor $\Phi \colon \mathbf{Sign} \to \mathbf{Sign}'$;
- a (natural) family of maps $\alpha_\Sigma \colon \mathbf{Sen}(\Sigma) \to \mathbf{Sen}'(\Phi(\Sigma))$, and
- a (natural) family of functors $\beta_\Sigma \colon \mathbf{Mod}'(\Phi(\Sigma)) \to \mathbf{Mod}(\Sigma)$,

such that for any $\Sigma \in |\mathbf{Sign}|$, any $\varphi \in \mathbf{Sen}(\Sigma)$ and any $M' \in \mathbf{Mod}'(\Phi(\Sigma))$:

$$M' \models'_{\Phi(\Sigma)} \alpha_\Sigma(\varphi) \textit{ iff } \beta_\Sigma(M') \models_\Sigma \varphi$$

$$[\textit{Satisfaction condition}]$$

Example comorphism: Prop to CASL

Translation of signatures: $\Phi(\Sigma) = (S, F, P)$ with

- sorts: $S = \emptyset$
- function symbols: $F_{w,s} = \emptyset$
- predicate symbols $P_w = \begin{cases} \Sigma, & \text{if } w = \lambda \\ \emptyset, & \text{otherwise} \end{cases}$.

Translation of sentences:

$$\alpha_\Sigma(\varphi) = \varphi$$

Translation of models: For $M' \in \mathrm{Mod}^{FOL}(\Phi(\Sigma))$ and $p \in \Sigma$ define

$$\beta_\Sigma(M')(p) := M'_p$$

The satisfaction condition is trivial.

Example comorphism: $\mathcal{ALC}$ to CASL

Translation of signatures:

$\Phi((\mathbf{C}, \mathbf{R}, \mathbf{I})) = (S, F, P)$ with

- sorts: $S = \{Thing\}$
- function symbols: $F = \{a\colon Thing \mid a \in \mathbf{I}\}$
- predicate symbols
  $P = \{A\colon Thing \mid A \in \mathbf{C}\} \cup \{R\colon Thing \times Thing \mid R \in \mathbf{R}\}$

Translation of concepts

Concepts are translated as follows (depending on some variable x):

- $\alpha_x(A) = A(x)$
- $\alpha_x(\top) = \top$
- $\alpha_x(\bot) = \bot$
- $\alpha_x(\neg C) = \neg\alpha_x(C)$
- $\alpha_x(C \sqcap D) = \alpha_x(C) \wedge \alpha_x(D)$
- $\alpha_x(C \sqcup D) = \alpha_x(C) \vee \alpha_x(D)$
- $\alpha_x(\exists R.C) = \exists y : \mathit{Thing}.(R(x,y) \wedge \alpha_y(C))$
- $\alpha_x(\forall R.C) = \forall y : \mathit{Thing}.(R(x,y) \rightarrow \alpha_y(C))$

Translation of sentences

- $\alpha_\Sigma(C \sqsubseteq D) = \forall x\colon \mathit{Thing}.\,(\alpha_x(C) \to \alpha_x(D))$
- $\alpha_\Sigma(a : C) = \alpha_x(C)[x \mapsto a]$[1]
- $\alpha_\Sigma(R(a, b)) = R(a, b)$

[1] $t[x \mapsto a]$ means "in t, replace x by a".

Translation of models

For $M' \in \mathrm{Mod}^{FOL}(\Phi(\Sigma))$ define $\beta_\Sigma(M') := \mathcal{I} := (\Delta, \cdot^{\mathcal{I}})$ with $\Delta = |M'|_{Thing}$ and $A^{\mathcal{I}} = M'_A$, $a^{\mathcal{I}} = M'_a$, $R^{\mathcal{I}} = M'_R$.

Lemma

$C^{\mathcal{I}} = \{m \in M'_{Thing} | M' + \{x \mapsto m\} \models \alpha_x(C)\}$

Proof.

By induction over the structure of C.

- $A^{\mathcal{I}} = M'_A = \{m \in M'_{Thing} | M' + \{x \mapsto m\} \models A(x)\}$
- $(\neg C)^{\mathcal{I}} = \Delta \setminus C^{\mathcal{I}}$
  $=^{I.H.} \Delta \setminus \{m \in M'_\top | M' + \{x \mapsto m\} \models \alpha_x(C)\}$
  $= \{m \in M'_\top | M' + \{x \mapsto m\} \models \neg\alpha_x(C)\}$ etc. $\square$

The satisfaction condition now follows easily.

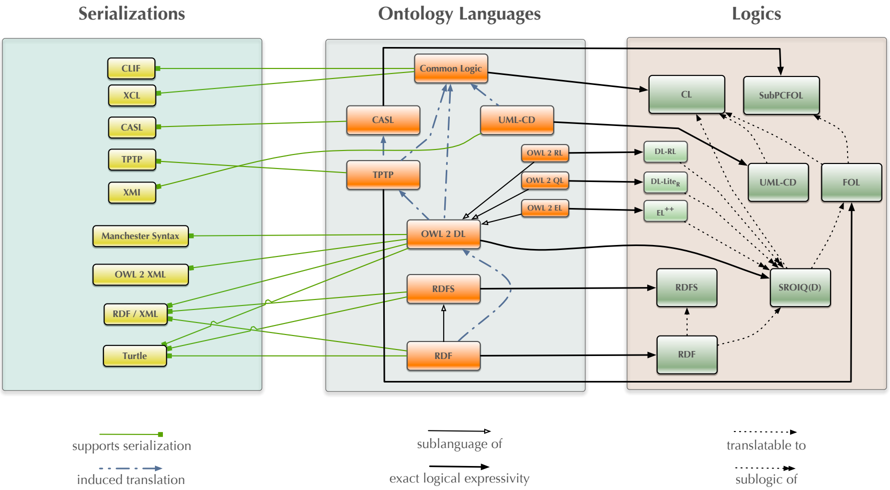
Heterogeneous logical environments

A heterogeneous logical environment ($\mathcal{HLE}$) consists of

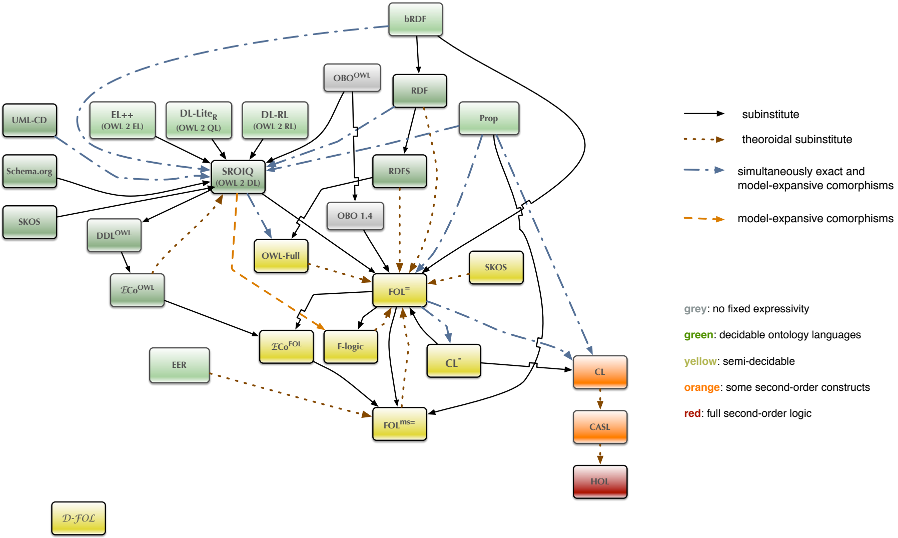
- a logic graph, consisting of institutions, institution comorphisms (translations) and institution morphisms (projections, see below),
- an OMS language graph, and
- support relations.

The support relations specify which language supports which logics and which serializations, and which language translation supports which logic translation or reduction.

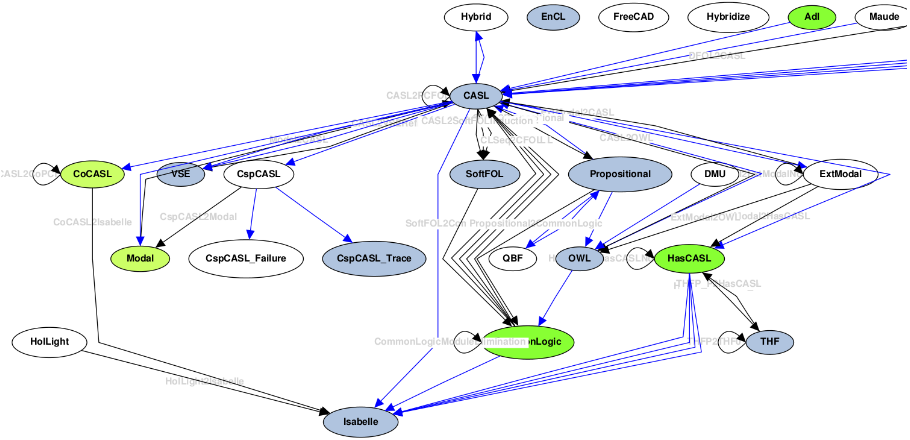
Moreover, for each language we have a default selection of a logic and a serialization. There are also default translations.


Serializations
Ontology Languages
Logics
CLIF
XCL
CASL
TPTP
XMI
Manchester Syntax
OWL 2 XML
RDF / XML
Turtle
Common Logic
CASL
UML-CD
TPTP
OWL 2 RL
OWL 2 QL
OWL 2 EL
OWL 2 DL
RDFS
RDF
CL
SubPCFOL
DL-RL
DL-Lite$_R$
EL^{++}
UML-CD
FOL
RDFS
SROIQ(D)
RDF
supports serialization
induced translation
sublanguage of
exact logical expressivity
translatable to
sublogic of

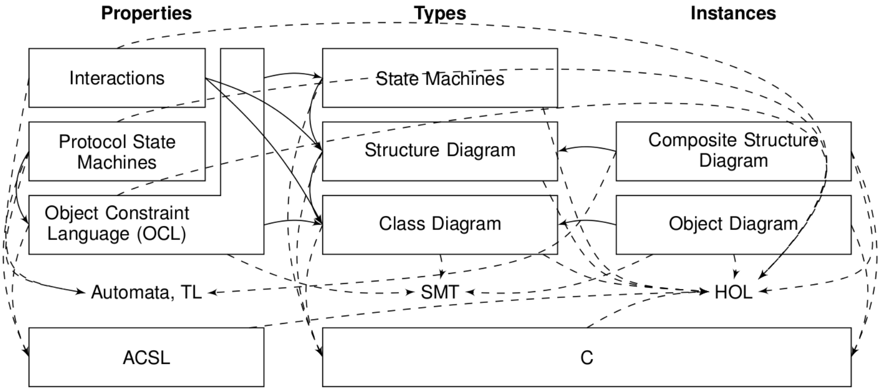
Ontologies: An Initial Logic Graph



Specifications: An Initial Logic Graph



UML models: An Initial Logic Graph



Heterogeneous Translations

Let ρ be an institution comorphism and O an OMS. Then we have the OMS

O with translation ρ

```
logic OWL
ontology Mereology =
  ObjectProperty: isPartOf
  ObjectProperty: isProperPartOf
   Characteristics: Asymmetric SubPropertyOf: isPartOf
  with translation OWL22CASL
then logic CASL :  {
  forall x,y,z:Thing .
     isProperPartOf(x,y) /\ isProperPartOf(y,z)
       => isProperPartOf (x,z) }
   %% transitivity; can't be expressed in OWL together
   %% with asymmetry
```

Semantic domains for OMS in DOL, revisited

Semantics of flattenable OMS (can be flattened to a basic OMS): (I, Σ, Ψ) (theory-level)

- I an institution
- Σ: a signature in I, also written $Sig(O)$
- Ψ: a set of Σ-sentences, also written $Th(O)$

Semantics of elusive OMS (= non-flattenable OMS): $(I, \Sigma, \mathcal{M})$ (model-level)

- I an institution
- Σ: a signature in I, also written $Sig(O)$
- $\mathcal{M}$: a class of Σ-models, also written $Mod(O)$

Semantics of heterogeneous translations

O flattenable Let $[\![O]\!]^T_\Gamma = (I, \Sigma, \Psi)$

- homogeneous translation
  $[\![O \textbf{ with } \sigma : \Sigma \to \Sigma']\!]^T_\Gamma = (I, \Sigma', \sigma(\Psi))$
- heterogeneous translation
  $[\![O \textbf{ with translation } \rho : I \to I']\!]^T_\Gamma = (I', \rho^{Sig}(\Sigma), \rho^{Sen}(\Psi))$

O elusive Let $[\![O]\!]^M_\Gamma = (I, \Sigma, \mathcal{M})$

- homogeneous translation
  $[\![O \textbf{ with } \sigma : \Sigma \to \Sigma']\!]^M_\Gamma = (I, \Sigma', \mathcal{M}')$
  where $\mathcal{M}' = \{M \in \textbf{Mod}(\Sigma') \mid M|_\sigma \in \mathcal{M}\}$
- heterogeneous translation
  $[\![O \textbf{ with translation } \rho : I \to I']\!]^M_\Gamma = (I', \rho^{Sig}(\Sigma), \mathcal{M}')$ where
  $\mathcal{M}' = \{M \in \textbf{Mod}^{I'}(\rho^{Sig}(\Sigma)) \mid \rho^{Mod}(M) \in \mathcal{M}\}$

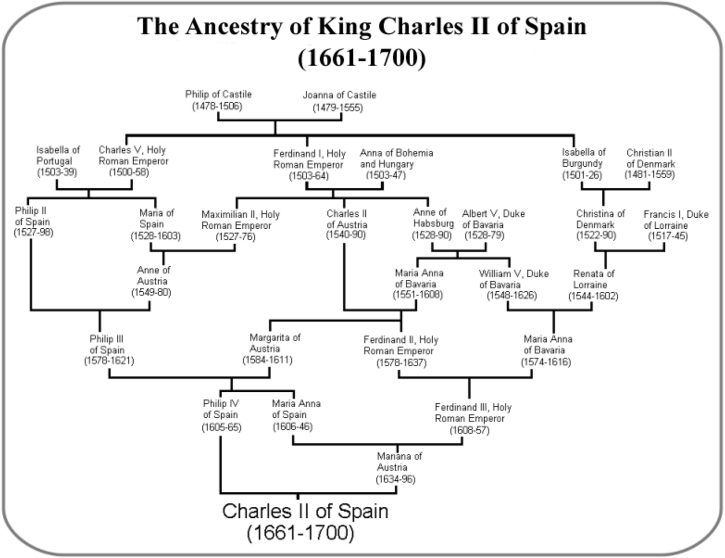
Extended task

New Task:

- Are there any inbreds people in our KB?

Charles II of Spain

What is an inbred? I



What is an inbred? II

u is inbread iff there are x y z such that

- x is a parent of u
- y is a parent of u
- $x \neq y$
- z is an ancestor of x
- z is an ancestor of y

Charles II of Spain

What is an inbred? II

u is inbread iff there are x y z such that

- x is a parent of u
- y is a parent of u
- $x \neq y$
- z is an ancestor of x
- z is an ancestor of y

DL has no variables $\rightarrow$ switch language

Charles II of Spain

Extended task: switch of logic

```
logic OWL
ontology Genealogy =
  ObjectProperty: parentOf SubPropertyOf: ancestor
  ObjectProperty: ancestor
  ObjectProperty: ancestor Characteristics: Transitive
end

ontology Inbred =
  Genealogy with translation OWL22CASL
then logic CASL :  {
  pred Inbred : Thing
  forall u:Thing
  . Inbred(u) <=> exists x,y,z:Thing .
       parentOf(x,u) /\ parentOf(y,u)
    /\ not x=y
    /\ ancestor(z,x) /\ ancestor(z,y) }
end
```

Extended task: entailment

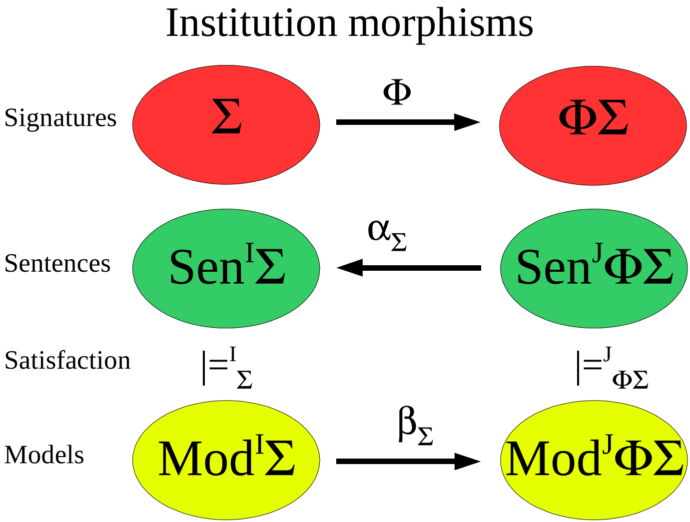
```
ontology CharlesII_ABox =
  Individual: CharlesII ... %% Charles II ABox
end

logic CASL
ontology anyInbreds =
  { CharlesII_ABox with translation OWL22CASL
  and Inbred }
then %implies
  . exists x:Thing . Inbred(x)
end
```

A heterogeneous reduction

```
ontology Inbred_OWL =
  Genealogy
and
logic CASL :  {
  sort Thing
  preds Inbred : Thing
        parentOf, ancestor : Thing*Thing
  forall u:Thing
  . Inbred(u) <=> exists x,y,z:Thing .
       parentOf(x,u)
    /\ parentOf(y,u)
    /\ not x=y
    /\ ancestor(z,x)
    /\ ancestor(z,y) } hide along OWL22CASL
end
```

This ontology imports first-order axioms only “on-the-fly”. Overall, it stays an OWL ontology (in contrast to the Inbred ontology).

Institution morphisms (projections)



Institution morphisms (projections)

Definition

Let $\mathcal{I} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ and $\mathcal{I}' = \langle \mathbf{Sign}', \mathbf{Sen}', \mathbf{Mod}', \langle \models'_{\Sigma'} \rangle_{\Sigma' \in |\mathbf{Sign}'|} \rangle$ be institutions. An institution morphism $\mu\colon \mathcal{I} \to \mathcal{I}'$ consists of:

- a functor $\mu^{Sign}\colon \mathbf{Sign} \to \mathbf{Sign}'$;
- a natural transformation $\mu^{Sen}\colon \mu^{Sign} ; \mathbf{Sen}' \to \mathbf{Sen}$, and
- a natural transformation $\mu^{Mod}\colon \mathbf{Mod} \to (\mu^{Sign})^{op} ; \mathbf{Mod}'$,

such that for any signature $\Sigma \in |\mathbf{Sign}|$, any $\varphi' \in \mathbf{Sen}'(\mu^{Sign}(\Sigma))$ and any $M \in \mathbf{Mod}(\Sigma)$:

$$M \models_\Sigma \mu^{Sen}_\Sigma(\varphi') \textit{ iff } \mu^{Mod}_\Sigma(M) \models'_{\mu^{Sign}(\Sigma)} \varphi'$$

$$[\mathit{Satisfaction\ condition}]$$

Example morphism: CASL to Prop

Translation of signatures: $\Phi((S, F, P)) = P_\lambda$.

Translation of sentences:

$$\alpha_\Sigma(\varphi) = \varphi$$

Translation of models: For $M' \in \mathrm{Mod}^{FOL}(\Phi(\Sigma))$ and $p \in \Sigma$ define

$$\beta_\Sigma(M')(p) := M'_p$$

The satisfaction condition is trivial.

Example morphism: single-sorted CASL to $\mathcal{ALC}$

Translation of signatures:

$\Phi((\{s\}, F, P)) = (\mathbf{C}, \mathbf{R}, \mathbf{I})$ with

- concepts: $\mathbf{C} = \{C \mid C\colon s \in P\}$
- roles: $\mathbf{R} = \{R \mid R\colon s \times s \in P\}$
- individuals $\mathbf{I} = \{a \mid a\colon s \in F\}$

Translation of sentences and models:

same as for the comorphism $\mathcal{ALC} \rightarrow$CASL.
Also the satisfaction condition follows in the same way.

Semantics of (heterogeneous) reductions

Let $[\![O]\!]_\Gamma^M = (I, \Sigma, \mathcal{M})$

- homogeneous reduction
  $[\![O \textbf{ reveal } \Sigma'\,]\!]_\Gamma^M = (I, \Sigma', \mathcal{M}|_{\Sigma'})$
  $[\![O \textbf{ hide } \Sigma'\,]\!]_\Gamma^M = [\![O \textbf{ reveal } \Sigma \setminus \Sigma']\!]_\Gamma^M$
- heterogeneous reduction
  $[\![O \textbf{ hide along } \rho : I \to I']\!]_\Gamma^M = (I', \rho^{Sig}(\Sigma), \rho^{Mod}(\mathcal{M}))$

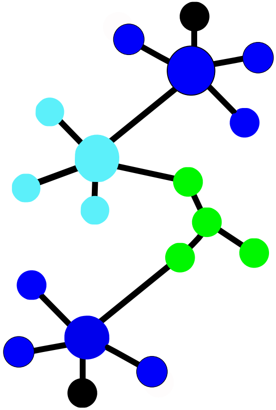
$\mathcal{M}|_{\Sigma'}$ may be impossible to capture by a theory (even if $\mathcal{M}$ is).

Semantics of heterogeneous approximation

Note: O must be flattenable!
Let $[\![O]\!]_\Gamma^T = (I, \Sigma, \Psi)$.

- homogeneous approximation
  $[\![O \textbf{ keep in } \Sigma']\!]_\Gamma^T = (I, \Sigma', \{\varphi \in \mathbf{Sen}(\Sigma') \mid \Psi \models \varphi\})$
  (note: any logically equivalent theory will also do)
  $[\![O \textbf{ forget } \Sigma']\!]_\Gamma^T = [\![O \textbf{ keep in } \Sigma \setminus \Sigma']\!]_\Gamma^T$
- heterogeneous approximation
  $[\![O \textbf{ keep in } \Sigma' \textbf{ with } I']\!]_\Gamma^T =$
  $(I', \Sigma', \{\varphi \in \mathbf{Sen}^{I'}(\Sigma') \mid \Psi \models \rho^{Sen}(\varphi)\})$
  where $\rho : I' \to I$ is the inclusion
  and Σ' is such that $\rho^{Sig}(\Sigma') \subseteq \Sigma$
  $[\![O \textbf{ forget } \Sigma' \textbf{ with } I']\!]_\Gamma^T = [\![O \textbf{ keep in } \Sigma \setminus \Sigma' \textbf{ with } I']\!]_\Gamma^T$

Networks and Their Combination



OMS networks (diagrams)

network N =
$N_1, \ldots, N_m, O_1, \ldots, O_n, M_1, \ldots, M_p$
excluding $N'_1, \ldots, N'_i, O'_1, \ldots, O'_j, M'_1, \ldots, M'_k$

- N_i are other networks
- O_i are OMS (possibly prefixed with labels, like $n : O$)
- M_i are mappings (views, interpretations)

Combinations

- combine N
- N is a network
- semantics is the (a) colimit of the diagram N

```
ontology AlignedOntology1 =
  combine N
```

Sample combination

```
ontology Source =
  Class: Person
  Class: Woman SubClassOf: Person
ontology Onto1 =
  Class: Person          Class: Bank
  Class: Woman SubClassOf: Person
interpretation I1 : Source to Onto1 =
   Person |-> Person, Woman |-> Woman
ontology Onto2 =
  Class: HumanBeing      Class: Bank
  Class: Woman SubClassOf: HumanBeing
interpretation I2 : Source to Onto2 =
   Person |-> HumanBeing, Woman |-> Woman
ontology CombinedOntology =
  combine Source, Onto1, Onto2, I1, I2
```

Resulting colimit

$$
\begin{tikzcd}
 & \{Woman, Person\_HumanBeing, 1 : Bank, 2 : Bank\} & \\
Onto1 \arrow[ur, dotted] & & Onto2 \arrow[ul, dotted] \\
 & \{Woman, Person\} \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

Alignments

- **alignment** $Id\ card_1\ card_2 : O_1$ **to** $O_2 = c_1, \ldots c_n$ **assuming** SingleDomain | GlobalDomain | ContextualizedDomain
- $card_i$ is (optionally) one of 1, ?, $+$, *
- the c_i are correspondences of form $sym_1\ rel\ conf\ sym_2$
  - sym_i is a symbol from O_i
  - rel is one of $>$, $<$, $=$, %, $\ni$, $\in$, $\mapsto$, or an Id
  - $conf$ is an (optional) confidence value between 0 and 1

Syntax of alignments follows the alignment API
`http://alignapi.gforge.inria.fr`

```
alignment Alignment1 : { Class: Woman } to { Class: Person } =
  Woman < Person
end
```

Alignment: Example

```
ontology S =  Class: Person
  Individual: alex Types: Person
  Class: Child

ontology T =  Class: HumanBeing
  Class: Male SubClassOf: HumanBeing
  Class: Employee

alignment A : S to T =
  Person = HumanBeing
  alex in Male
  Child < not Employee
  assuming GlobalDomain
```

Networks, revisited

network N =
$N_1, \ldots, N_m, O_1, \ldots, O_n, M_1, \ldots, M_p, A_1, \ldots, A_r$
excluding $N'_1, \ldots, N'_i, O'_1, \ldots, O'_j, M'_1, \ldots, M'_k$

- N_i are other networks
- O_i are OMS (possibly prefixed with labels, like $n : O$)
- M_i are mappings (views, equivalences)
- A_i are alignments

The resulting diagram N includes (institution-specific) W-alignment diagrams for each alignment A_i. Using **assuming**, assumptions about the domains of all OMS can be specified:

SingleDomain aligned symbols are mapped to each other

GlobalDomain aligned OMS are relativized

ContextualizedDomain alignments are reified as binary relations

Diagram of a SingleDomain alignment

$$
\begin{tikzcd}
S & & B & & T \\
& \{Person, alex, Child\} \arrow[ul, "\iota_1"] \arrow[ur, "\sigma_1"'] & & \{Male, HumanBeing, Employee\} \arrow[ul, "\sigma_2"] \arrow[ur, "\iota_2"'] &
\end{tikzcd}
$$

where

ontology B =

Class: *Person_HumanBeing*
Class: *Employee*
Class: *Child*
SubClassOf: ¬ *Employee*
Individual: *alex*
Types: *Male*

Resulting colimit

The colimit ontology of the diagram of the alignment above is:

ontology B = **Class:** *Person_HumanBeing*
Class: *Employee*
Class: *Male* **SubClassOf:** *Person_HumanBeing*
Class: *Child* **SubClassOf:** ¬ *Employee*
Individual: *alex* **Types:** *Male*, *Person_HumanBeing*

Background: Simple semantics of diagrams

Framework: institutions like OWL, FOL, . . .
OMS are interpreted over the same domain

$$\begin{tikzcd}
O_1 \arrow[drr, "m_1"'] & O_2 \arrow[dr, "m_2"] & \dots & & O_n \arrow[dll, "m_n"] \\
& & D & &
\end{tikzcd}$$

- model for A: (m_1, m_2) such that $m_1(s)\ R\ m_2(t)$ for each $s\ R\ t$ in A
- model for a diagram: family (m_i) of models such that (m_i, m_j) is a model for A_{ij}
- local models of O_j modulo a diagram: jth-projection on models of the diagram

Alignment of Bioportal Ontologies

```
logic OWL
%prefix(
  ontologies: <https://ontohub.org/bioportal/>
  obo: <http://purl.obolibrary.org/obo/> )%
alignment ZFA2MA : ontologies:ZFA to ontologies:MA =
%% ZFA: zebrafish anatomical ontology
%% MA: adult mouse anatomy
  obo:ZFA_0005153 = obo:MA_0000322,
  obo:ZFA_0001197 = obo:MA_0000855,
  obo:ZFA_0000529 = obo:MA_0000368,
  obo:ZFA_0000413 = obo:MA_0002420,
  obo:ZFA_0000816 = obo:MA_0000344,
  obo:ZFA_0001114 = obo:MA_0000023,
  obo:ZFA_0000010 = obo:MA_0000010,
  obo:ZFA_0000539 = obo:MA_0001017,
  obo:ZFA_0001101 = obo:MA_0002446  end
ontology combination = %cons
  combine ZFA2MA   end
```

Alignment of Bioportal Ontologies

```
logic OWL
%prefix(
  ontologies: <https://ontohub.org/bioportal/>
  obo: <http://purl.obolibrary.org/obo/> )%
alignment ZFA2MA : ontologies:ZFA to ontologies:MA =
%% ZFA: zebrafish anatomical ontology
%% MA: adult mouse anatomy
  obo:synovial joint = obo:synovial joint,
  obo:pars intermedia = obo:pars intermedia,
  obo:kidney = obo:kidney,
  obo:gonad = obo:gonad,
  obo:oral epithelium = obo:oral epithelium,
  obo:head = obo:head,
  obo:cardiovascular system = obo:cardiovascular system,
  obo:locus coeruleus = obo:locus coeruleus,
  obo:gustatory system = obo:gustatory system  end
ontology combination = %cons
  combine ZFA2MA   end
```

Alignment of Upper Ontologies

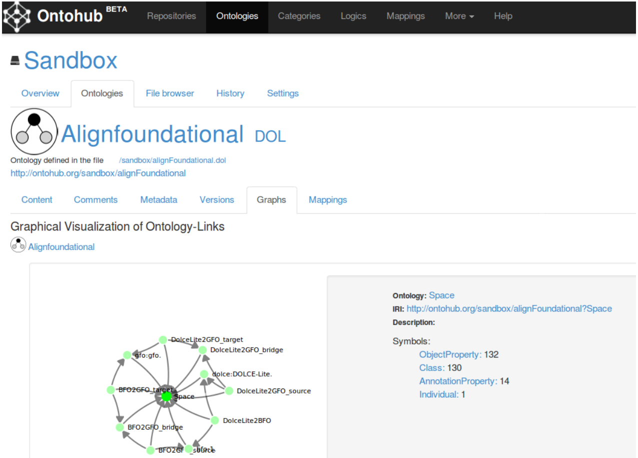
```
%prefix(   gfo: <http://www.onto-med.de/ontologies/>
           dolce: <http://www.loa-cnr.it/ontologies/>
           bfo: <http://www.ifomis.org/bfo/>            )%

logic OWL

alignment DolceLite2BFO :  dolce:DOLCE-Lite.owl to bfo:1.1 =
 endurant = IndependentContinuant,
 physical-endurant = MaterialEntity,
 physical-object = Object,   perdurant = Occurrent,
 process = Process,          quality = Quality,
 spatio-temporal-region = SpatiotemporalRegion,
 temporal-region = TemporalRegion,  space-region = SpatialRegion
```

Alignment of Upper Ontologies (cont'd)

```
alignment DolceLite2GFO : dolce:DOLCE-Lite.owl to gfo:gfo.owl =
  particular = Individual, endurant = Presential,
  physical-object = Material_object,
  amount-of-matter = Amount_of_substrate,
  perdurant = Occurrent,  quality = Property,
  time-interval = Chronoid, generic-dependent < necessary_for,
  part < abstract_has_part, part-of < abstract_part_of,
  proper-part < has_proper_part,
  proper-part-of < proper_part_of,
  generic-location < occupies,
  generic-location-of < occupied_by

alignment BFO2GFO :  bfo:1.1 to gfo:gfo.owl =
 Entity = Entity, Object = Material_object,
 ObjectBoundary  = Material_boundary, Role < Role ,
 Occurrent = Occurrent,  Process = Process, Quality = Property
 SpatialRegion = Spatial_region,
 TemporalRegion = Temporal_region
```

Alignment of Upper Ontologies — Combination

```
ontology Space =
  combine BFO2GFO, DolceLite2GFO, DolceLite2BFO
```



Integrated semantics of diagrams

Framework: different domains reconciled in a global domain

$$\begin{tikzcd}
O_1 \arrow[d, "m"'] & O_2 \arrow[d, "m_2"] & \dots & O_n \arrow[d, "m_n"] \\
D_1 \arrow[drr, "\gamma_1"'] & D_2 \arrow[dr, "\gamma_2"] & \dots & D_n \arrow[dl, "\gamma_n"] \\
& & U &
\end{tikzcd}$$

- model for a diagram: family (m_i) of models with equalizing function γ such that $(\gamma_i m_i, \gamma_j m_j)$ is a model for A_{ij}

Relativization of an OWL ontology

Let O be an ontology, define its relativization $\tilde{O}$:

- concepts are concepts of O with a new concept $\top_O$;
- roles and individuals are the same
- axioms:
  - each concept C is subsumed by $\top_O$,
  - each individual i is an instance of $\top_O$,
  - each role r has domain and range $\top_O$.

  and the axioms of O where the following replacement of concept is made:
  - each occurence of $\top$ is replaced by $\top_O$,
  - each concept $\neg C$ is replaced by $\top_O \setminus C$, and
  - each concept $\forall R.C$ is replaced by $\top_O \sqcap \forall R.C$.

Example: integrated semantics

$$\begin{tikzcd}
 & & C & & \\
\tilde{S} \arrow[urr, dotted] & & B \arrow[u, dotted] & & \tilde{T} \arrow[ull, dotted] \\
 & \{Person, alex, Child\} \arrow[ul, "\iota_1"] \arrow[ur, "\sigma_1"'] \arrow[uur, dotted] & & \{Male, HumanBeing, Employee\} \arrow[ul, "\sigma_2"] \arrow[ur, "\iota_2"'] \arrow[uul, dotted] &
\end{tikzcd}$$

where

ontology B =

Class: $Thing_S$ **Class:** $Thing_T$
Class: *Person_HumanBeing* **SubClassOf:** $Thing_S$, $Thing_T$
Class: *Male* **Class:** *Employee*
Class: *Child* **SubClassOf:** $Thing_T$ **and** $\neg$ *Employee*
Individual: *alex* **Types:** *Male*

Example: integrated semantics (cont'd)

ontology C =
Class: *ThingS*
Class: *ThingT*
Class: *Person_HumanBeing* **SubClassOf:** *ThingS*, *ThingC*
Class: *Male* **SubClassOf:** *Person_HumanBeing*
Class: *Employee* **SubClassOf:** *ThingT*
Class: *Child* **SubClassOf:** *ThingS*
Class: *Child* **SubClassOf:** *ThingT* **and** ¬ *Employee*
Individual: *alex* **Types:** *Male*, *Person_HumanBeing*

Contextualized semantics of diagrams

Framework: different domains related by coherent relations

$$
\begin{tikzcd}
O_1 \arrow[d, "m"'] & O_2 \arrow[d, "m_2"] & \dots & O_n \arrow[d, "m_n"] \\
D_1 \arrow[r, "r_{1,2}"] \arrow[rr, bend right, "r_{1,3}"] & D_2 \arrow[r, "r_{2,3}"] & \dots & D_n \arrow[lll, bend left, "r_{n,1}"]
\end{tikzcd}
$$

such that

- r_{ij} is functional and injective,
- r_{ii} is the identity (diagonal) relation,
- r_{ji} is the converse of r_{ij}, and
- r_{ik} is the relational composition of r_{ij} and r_{jk}
- model for a diagram: family (m_i) of models with coherent relations (r_{ij}) such that $(m_i, r_{ji} m_j)$ is a model for A_{ij}

Contextualized semantics of diagrams, revisited

$$\begin{tikzcd}
\tilde{S} & & \overline{B} & & \tilde{T} \\
& O_1 \arrow[ul, "\iota_1"'] \arrow[ur, "\sigma_1"] & & O_2 \arrow[ul, "\sigma_2"'] \arrow[ur, "\iota_2"] &
\end{tikzcd}$$

where $\overline{B}$ modifies B as follows:

- r_{ij} are added to $\overline{B}$ as roles with domain $\top_S$ and range $\top_T$
- the correspondences are translated to axioms involving these roles:
  - $s_i = t_j$ becomes $s_i \ r_{ij} \ t_j$
  - $a_i \in c_j$ becomes $a_i \in \exists r_{ij}.c_j$
  - ...
- the properties of the roles are added as axioms in $\overline{B}$

Adding domain relations to the bridge

ontology $\overline{B} =$

Class: *ThingS*
Class: *ThingT*
ObjectPropery: r_{ST} **Domain:** *ThingS* **Range:** *ThingT*
Class: *Person* **EquivalentTo:** r_{ST} **some** *HumanBeing*
Class: *Employee*
Class: *Child* **SubClassOf:** r_{ST} **some** $\neg$ *Employee*
Individual: *alex* **Types:** r_{ST} **some** *Male*

Example: contextualized semantics

$$
\begin{tikzcd}
 & & C & & \\
S \arrow[urr, dotted] & & \overline{B} \arrow[u, dotted] & & T \arrow[ull, dotted] \\
 & \{Person, alex, Child, \top_s\} \arrow[ul, "\iota_1"] \arrow[ur, "\sigma_1"'] \arrow[uur, dotted] & & \{Male, HumanBeing, Employee, \top_T\} \arrow[ul, "\sigma_2"] \arrow[ur, "\iota_2"'] \arrow[uul, dotted] &
\end{tikzcd}
$$

where

ontology C =

Class: *ThingS*
Class: *ThingT*
ObjectPropery: r_{ST} **Domain:** *ThingS* **Range:** *ThingT*
Class: *Person* **EquivalentTo:** r_{ST} **some** *HumanBeing*
Class: *Employee*
Class: *Child* **SubClassOf:** r_{ST} **some** $\neg$ *Employee*
Individual: *alex* **Types:** r_{ST} **some** *Male*, *Person*

Refinements

$$O_1 \rightsquigarrow O_2$$

Recall Sorting Example

Informal specification:

To sort a list means to find a list with the same elements, which is in ascending order.

Formal requirements specification:

```
%right_assoc( __::__ )%
logic CASL.FOL=
spec PartialOrder =
  sort Elem
  pred __leq__ : Elem * Elem
  . forall x : Elem . x leq x %(refl)%
  . forall x, y : Elem . x leq y /\ y leq x => x = y %(antisym)%
  . forall x, y, z : Elem . x leq y /\ y leq z => x leq z %(trans)%
end
spec List =  PartialOrder then
  free type List ::= [] | __::__(Elem; List)
  pred __elem__ : Elem * List
  forall x,y:Elem; L,L1,L2:List
  . not x elem []
  . x elem (y :: L) <=>  x=y \/ x elem L
end
```

Sorting (cont'd)

```
spec AbstractSort =
  List
then %def
  preds is_ordered : List;
        permutation : List * List
  op sorter : List->List
  forall x,y:Elem; L,L1,L2:List
  . is_ordered([])
  . is_ordered(x::[])
  . is_ordered(x::y::L) <=> x leq y /\ is_ordered(y::L)
  . permutation(L1,L2) <=>
          (forall x:Elem . x elem L1 <=> x elem L2)
  . is_ordered(sorter(L))
  . permutation(L,sorter(L))
end
```

Sorting (cont'd)

We want to show insert sort to enjoy these properties.
Formal design specification:

```
spec InsertSort =  List then
  ops insert : Elem*List -> List;
      insert_sort : List->List
  vars x,y:Elem; L:List
  . insert(x,[]) = x::[]
  . x leq y => insert(x,y::L) = x::insert(y,L)
  . not x leq y => insert(x,y::L) = y::insert(x,L)
  . insert_sort([]) = []
  . insert_sort(x::L) = insert(x,insert_sort(L))
end
```

Implementation (in Haskell)

```
spec HaskellInsertSort =
insert :: Ord a => (a,[a]) -> [a]
insert(x,[]) = [x]
insert(x,y:l) = if x <= y then x:y:l
                  else y:insert(x,l)

insert_sort :: Ord a => [a] -> [a]
insert_sort([]) = []
insert_sort(x:l) = insert(x,insert_sort(l))
end
```

Refinement

We have the following refinement steps:
AbstractSort $\rightsquigarrow$ InsertSort $\rightsquigarrow$ HaskellInsertSort

```
refinement R =
  AbstractSort
     refined to InsertSort
     refined via CASL2Haskell to HaskellInsertSort
end
```

Refinement of Natural Numbers

```
spec Monoid =
 sort Elem
 ops 0 : Elem;
        __+__ : Elem * Elem -> Elem, assoc, unit 0
end
spec NatWithSuc = %mono
 free type Nat ::= 0 | suc(Nat)
 op __+__ : Nat * Nat -> Nat, unit 0
 forall x , y : Nat . x + suc(y) = suc(x + y)
 op 1:Nat = suc(0)
end
spec Nat =
  NatWithSuc hide suc
end
```

Refinement of Natural Numbers (cont'd)

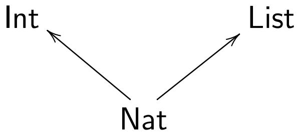
```
spec NatBin =
generated type Bin ::= 0 | 1 | __0(Bin) | __1(Bin)
ops __+__ , __++__ : Bin * Bin -> Bin
forall x, y : Bin
 .  0 0 = 0  .  0 1 = 1
 .  not  (0 = 1)  .  x 0 = y 0 => x = y .  not  (x 0 = y 1)  .
x 1 = y 1 => x = y
 .  0 + 0 = 0  .  0 ++ 0 = 1
 .  x 0 + y 0 = (x + y) 0  .  x 0 ++ y 0 = (x + y) 1
 .  x 0 + y 1 = (x + y) 1  .  x 0 ++ y 1 = (x ++ y) 0
 .  x 1 + y 0 = (x + y) 1  .  x 1 ++ y 0 = (x ++ y) 0
 .  x 1 + y 1 = (x ++ y) 0  .  x 1 ++ y 1 = (x ++ y) 1
end
```

Refinement of Natural Numbers (cont'd)

```
refinement R1 =
 Monoid refined via sort Elem |-> Nat to Nat
end
refinement R2 =
 Nat refined via sort Nat |-> Bin to NatBin
end
refinement R3 =
 Monoid refined via sort Elem |-> Nat to
 Nat refined via sort Nat |-> Bin to NatBin
end
refinement R3' =
 Monoid refined via sort Elem |-> Nat to R2
end
refinement R3'' =
 Monoid refined via sort Elem |-> Nat to Nat then R2
end
refinement R3''' = R1 then R2
```

Sample Network

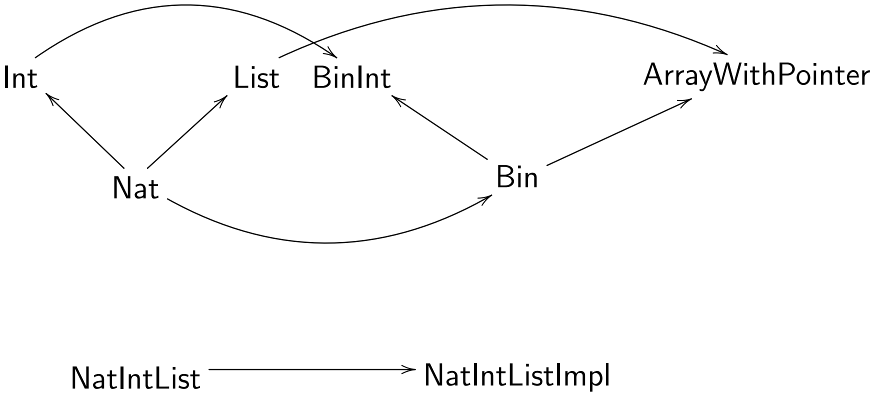
```
spec Nat = ...
end
spec Int = Nat then ...
end
spec List = Nat then ...
end
network NatIntList = Nat, Int, List
end
```



Sample Refinement of Networks

```
spec NatBin = ...
end
spec IntBin = NatBin then ...
end
spec ArrayWithPointer = NatBin then ...
end
network NatIntListImpl = NatBin, IntBin, ArrayWithPointer
end
refinement NetRefine =
  NatIntList refined via
      R2,
      Int refined via sort Int |-> BinInt to IntBin,
      List via sort List |-> Array to ArrayWithPointer
    to NatIntListImpl
end
```

The Refinement, Graphically



Entailments, Equivalences, Queries

Entailments

- entailment $Id = O_1$ entails O_2
- use case: Ontology **entails** competency questions

```
entailment e =
  BFO_FOL entails { BFO_OWL with translation OWL2FOL }
end
```

Equivalences

- **equivalence** $Id : O_1 \leftrightarrow O_2 = O_3$
- (fragment) OMS O_3 is such that O_i **then %def** O_3 is a definitional extension of O_i for $i = 1, 2$;
- this implies that O_1 and O_2 have model classes that are in bijective correspondence

```
equivalence e : algebra:BooleanAlgebra
                ↔ algebra:BooleanRing =
    x∧y = x·y
    x∨y = x+y+x·y
    ¬x = 1+x
    x·y = x∧y
    x+y = (x∨y) ∧ ¬(x∧y)
end
```

Conservativity Definitions (Module Relations)

- **cons-ext** $Id\ c : O_1$ **of** O_2 **for** Σ
- O_1 is a module of O_2 with restriction signature Σ and conservativity c

c=%mcons	every Σ-reduct of an O_1-model can be expanded to an O_2-model
c=%ccons	every Σ-sentence φ following from O_2 already follows from O_1

This relation shall hold for any module O_1 extracted from O_2 using the **extract** construct.

Queries

DOL is a logical (meta) language

- focus on ontologies, models, specifications,
- and their logical relations: logical consequence, interpretations, ...

Queries are different:

- answer is not "yes" or "no", but an answer substitution
- query language may differ from language of OMS that is queried

Sample query languages

- conjunctive queries in OWL
- Prolog/Logic Programming
- SPARQL

Tentative Proposal for Syntax of Queries in DOL

New OMS declarations and relations:

```
query qname = select vars where sentence in OMS
              [along language-translation]
substitution sname : OMS1 to OMS2 = derived-symbol-map
result rname = sname_1, ..., sname_n for qname
             %% result is a substitution
```

New sentences (however, as structured OMS!):

```
apply(sname,sentence)    %% apply substition
```

Open question: how to deal with “construct” queries?

Conclusion

Conclusion

- DOL is a meta language for (formal) ontologies, specifications and models (OMS)
- DOL covers many aspects of modularity of and relations among OMS ("OMS-in-the large")
- DOL is standardized at OMG
- you can help with joining the DOL discussion
  - see `dol-omg.org`

Challenges

- What is a suitable abstract meta framework for non-monotonic logics and rule languages like RIF and RuleML? Are institutions suitable here? different from those for OWL?
- What is a useful abstract notion of query (language) and answer substitution?
- How to integrate TBox-like and ABox-like OMS?
- Can the notions of class hierarchy and of satisfiability of a class be generalised from OWL to other languages?
- How to interpret alignment correspondences with confidence other that 1 in a combination?
- Can logical frameworks be used for the specification of OMS languages and translations?
- Proof support for all of DOL

Thank you for your attention

In case of questions, contact us:

Oliver Kutz
Oliver.Kutz@unibz.it

Till Mossakowski
till@iks.cs.ovgu.de