The Distributed Ontology, Model and Specification Language (DOL)
Day 1: Motivation and Introduction

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

Welcome to DOL!

Lectures:

- Day 1: Motivation and Introduction
- Day 2: Basic Structuring with DOL
- Day 3: Structured OMS and Their Semantics
- Day 4: Using Multiple Logical Systems
- Day 5: Advanced Concepts and Applications

Welcome to DOL!

Daily practical sessions:

- We will learn the basics of how to use DOL in practice employing the `Ontohub.org` platform and the HETS.eu proof management and reasoning system.

Background:

DOL is for:

1. Ontology engineering (e.g. working with OWL or FOL)
2. Model-driven engineering (e.g. working with UML, ORM)
3. Formal (algebraic) specification (e.g. working with FOL, CASL, VDM, Z)

DOL is a metalanguage providing formal syntax & semantics for all of them!

Motivation from ontology engeneering:

We begin with the question:

- What kind of **ontology** engineering problems does DOL address?

Note:

- The issues/problems disscussed in the following apply equally to **model-driven engineering** and **formal specification**, and to other uses of logical theories.

Examples throughout the course will be taken from the ontology world (understood as logical theories), using propositional, description, and first-order logic, but also from algebra, mereotopology, and software specification.

Where we are in the ontology landscape

- Formal ontology
- Ontology based on linguistic observations
- Ontology based on scientific evidence
- Ontology as information system
- Ontology languages

A basic problem in ontology engineering:

How can we make it easier to build better ontologies?

A basic problem in ontology engineering:

How can we make it easier to build better ontologies?

Claim:
Distributed Ontology, Model and Specification Language (DOL) solves many basic (and advanced) ontology engineering problems

Assume you need to build an ontology

Three challenges for aspiring ontologist

1. Reuse of ontologies
2. Diversity of languages
3. Evaluate against requirements

Three challenges for aspiring ontologist

1. Reuse of ontologies
2. Diversity of languages
3. Evaluate against requirements

Reuse of ontologies I

First idea:
Reuse existing resources

Reuse of ontologies II

Reuse is hard

- Terminology is "wrong"
- Ontology is too wide
- Different ontologies pieces don't fit to each other

Reuse of ontologies II

Reuse is hard

- Terminology is "wrong"
- Ontology is too wide
- Different ontologies pieces don't fit to each other

Modifying local copies of ontologies leads to maintenance issues

Three challenges for aspiring ontologist

1. Reuse of ontologies
2. Diversity of languages
3. Evaluate against requirements

Diversity of OMS Languages

Languages that have been used for ontological modelling:

- First-order logic
- Higher-order logic
- OWL (Lite, EL, QL, RL, DL, Full), other DLs
- UML (e.g. class diagrams)
- Entity Relationship Diagrams
- Other languages: SWRL, RIF, ORM, BPMN, . . .

Which language should I use?

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.

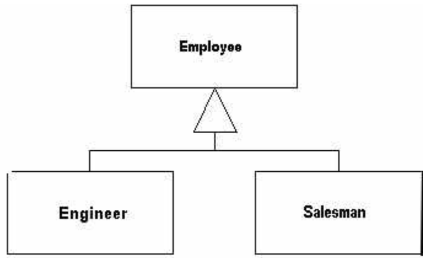
[2] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Descriptve ontology for linguistic and cognitive engineering. `http://www.loa.istc.cnr.it/DOLCE.html`.

Challenge for combined ontologies I: Where is the glue?

- The different modules need to be fitted together.
- Challenge: Languages may differ widely with respect to syntactic categories!

Challenge for combined ontologies II: Consistency

- Different people work independently on different parts.
- How do we ensure consistency across the whole ontology?
- Automatic theorem provers are specialized in one language.



$\forall x \sim ((\text{Contractor}\, x) \wedge (\text{Employee}\, x))$
(bob : Contractor), (bob : Engineer)

Diversity of Language: Conclusion

Use of different languages

- theoretically good idea
- leads to interoperability problems
- obstacle to reuse of ontologies

Three challenges for aspiring ontologist

1. Reuse of ontologies
2. Diversity of languages
3. Evaluate against requirements

Frequently asked question by students

Competency Questions – Simplified Summary

- Let O be an ontology
- Capture requirements for O as pairs of scenarios and competency questions
- For each scenario competency question pair S, Q:
  - Formalize S, resulting in theory Γ
  - Formalize Q, resulting in formula φ
  - Check with theorem prover whether $O \cup \Gamma \vdash \varphi$
- When all proofs are successful, your ontology meets the requirements.

Competency Questions Revisited

- CQ most successful idea for ontology evaluation
- Technically, CQ = proof obligations
- Language for expressing proof obligations?
- Ad hoc handling of CQs

Competency Questions Challenge

- How do we keep track of scenarios and competency questions in a systematic way?

Competency Questions Challenge

- How do we keep track of scenarios and competency questions in a systematic way?

DOL provides a systematic solution to this: ⇒ Lecture 2

What does “Modifying / Reusing” mean?

- Translations between ontology languages
- Renaming of symbols
- Unions of ontologies
- Removing of axioms
- Module extraction
- ...

None of these features are directly supported by widely used languages such as OWL or FOL.

What does "Modifying / Reusing" mean?

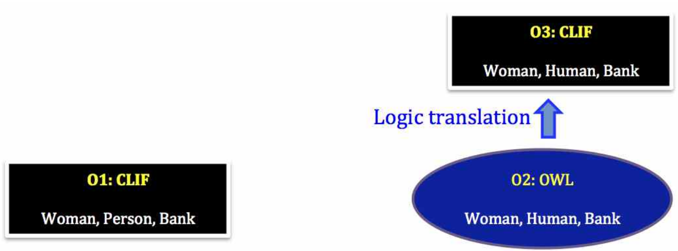
- Translations between ontology languages
- Renaming of symbols
- Unions of ontologies
- Removing of axioms
- Module extraction
- ...

None of these features are directly supported by widely used languages such as OWL or FOL.

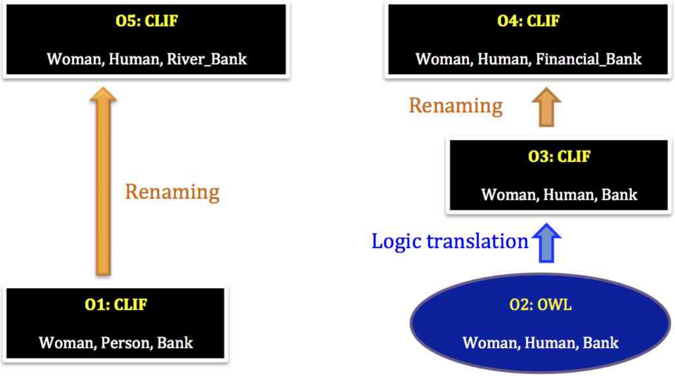
DOL covers all these operations: $\Rightarrow$ Lecture 2–4

Example Modifying / Reusing

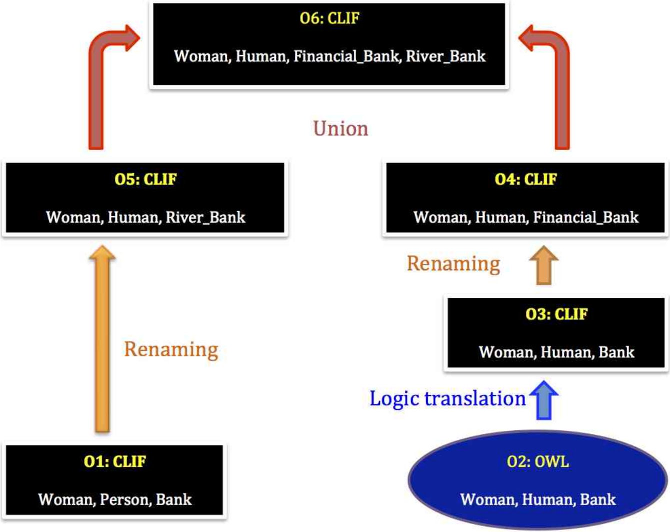
Example Modifying / Reusing



Example Modifying / Reusing



Example Modifying / Reusing



Declaration of Relations: Example Bridge Axiom

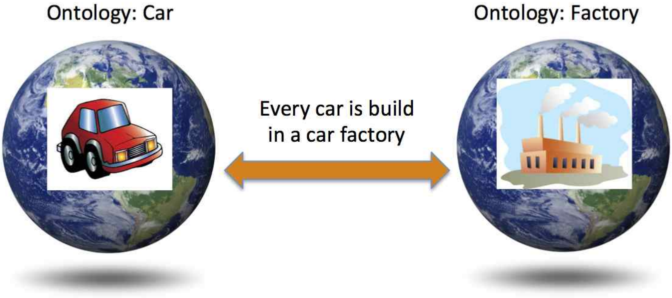
Ontology: Car

Declaration of Relations: Example Bridge Axiom

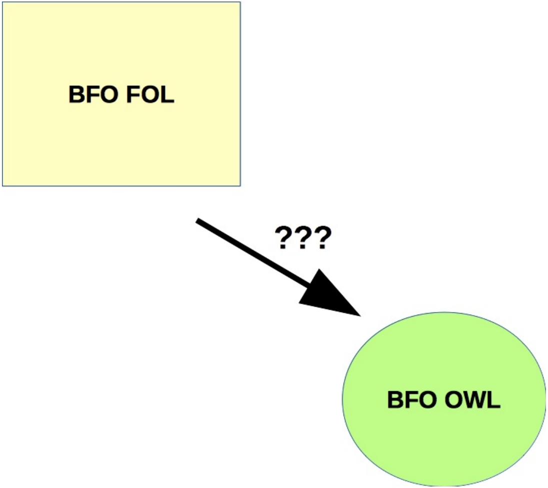
Ontology: Car

Ontology: Factory

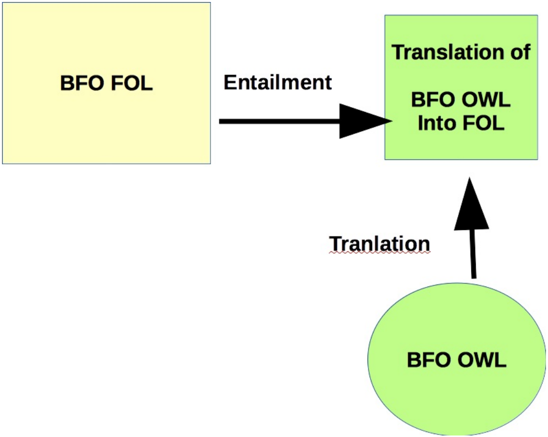
Declaration of Relations: Example Bridge Axiom



Specification of Intended Relations: Example BFO (Basic Formal Ontology)



Specification of Intended Relations: Example BFO (Basic Formal Ontology)



DOL: change in perspective

- Modular design vs ontology blobs

Ontologies are often big monolithic blobs

National Center for Biotechnology Information (NCBI) Organismal Classification (NCBITAXON)

projects: 12 | classes: 906,907

The NCBI Taxonomy Database is a curated classification and nomenclature for all of the organisms in the public sequence databases.

Uploaded: 6/10/15

The Drug Ontology (DRON)

classes: 408,573

An ontology of drugs

Uploaded: 5/2/15

Systematized Nomenclature of Medicine - Clinical Terms (SNOMEDCT)

notes: 2 | projects: 18 | classes: 316,031

SNOMED Clinical Terms

Uploaded: 6/10/15

Robert Hoehndorf Version of MeSH (RH-MESH)

projects: 3 | classes: 305,349

Medical Subjects Headings Thesaurus 2014, Modified version

Uploaded: 4/22/14

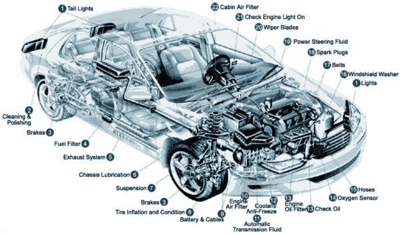
Cell Cycle Ontology (CCO)

projects: 2 | classes: 277,764

An application ontology integrating knowledge about the eukaryotic cell cycle.

Uploaded: 3/7/15

Engineers like it modular



Obvious benefits of modular design

Modularity allows for better

- Maintainability
- Reusability
- Quality control
- Adaptability

Obvious benefits of modular design

Modularity allows for better

- Maintainability
- Reusability
- Quality control
- Adaptability

Why not in ontology engineering?

The OMG standard DOL: Basic Ideas

DOL – An OMG standard

- DOL = Distributed Ontology, Model, and Specification Language
- OMG Specification, Beta 1 released
- Has been approved by OMG
- Now in finalization process

History of DOL

- First Initiative: Ontology Integration and Interoperability (OntoIOp)
- started in 2011 as ISO 17347 within ISO/TC 37/SC 3
- now continued as OMG standard
  - OMG has more experience with formal semantics
  - OMG documents will be freely available
  - focus extended from ontologies only to formal models and specifications (i.e. logical theories)
  - vote for DOL becoming a standard taken in Spring 2016
  - now finalization task force until end of 2016
- 50 experts participate, $\sim$ 15 have actively contributed
- DOL is open for your ideas, so join us!

The Big Picture of Interoperability

Modeling	Specification	Ontology engineering
Objects/data	Software	Concepts/data
Models	Specifications	Ontologies
Modeling Language	Specification language	Ontology language

Diversity and the need for interoperability occur at all these levels!

What have ontologies, models and specifications in common?

OMS ...

- are formalised in some logical system
- have a signature with non-logical symbols (domain vocabulary)
- have axioms expressing the domain-specific facts
- semantics: class of structures (models) interpreting signature symbols in some semantic domain
- we are interested in those structures (models) satisfying the axioms
- rich set of annotations and comments

In DOL, ontologies, models and specifications are called "OMS"!

DOL metalanguage capabilities

DOL enables reusability and interoperability.
DOL is a meta-language:

- Literally **reuse** existing OMS
- Operations for **modifying**/reusing OMS
- Declaration of **relations** between OMS
- Declaration of **intended relationships** between OMS
- Support for **heterogenous** OMS

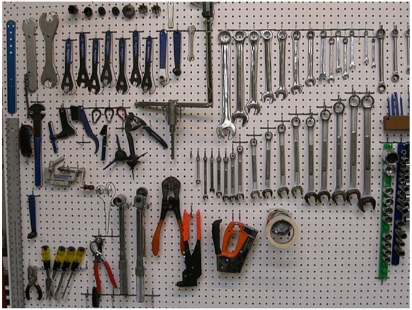
Diversity of Operations on and Relations among OMS

Various operations and relations on OMS are in use:

- structuring: import, union, translation, hiding, ...
- alignment
  - of many OMS covering one domain
- module extraction
  - get **relevant information** out of large OMS
- approximation
  - model in an **expressive** language, **reason fast** in a lightweight one
- distributed OMS
  - **bridges** between different modellings
- refinement / interpretation

From Babylonian Confusion to Toolkit

→



There is a Need for a Unifying Meta Language

Not yet another OMS language, but a meta language covering

- diversity of OMS languages
- translations between these
- diversity of operations on and relations among OMS

Current standards like the OWL API or the aligment API only cover parts of this

The DOL standard addresses this

The DOL language requires abstract semantics covering a diversity of OMSs.

Overview of DOL: Toolkit in Summary

1. OMS
   - basic OMS (flattenable)
   - references to named OMS
   - extensions, unions, translations (flattenable)
   - reductions, minimization, maximization (elusive)
   - approximations, module extractions, filterings (flattenable)
   - combinations of networks (flattenable)

   (flattenable = can be flattened to a basic OMS)
2. OMS mappings (between OMS)
   - interpretations, refinements, alignments, . . .
3. OMS networks (based on OMS and mappings)
4. OMS libraries (based on OMS, mappings, networks)
   - OMS definitions (giving a name to an OMS)
   - definitions of interpretations, refinements, alignments
   - definitions of networks, entailments, equivalences, . . .

DOL Semantic Foundations: Institutions

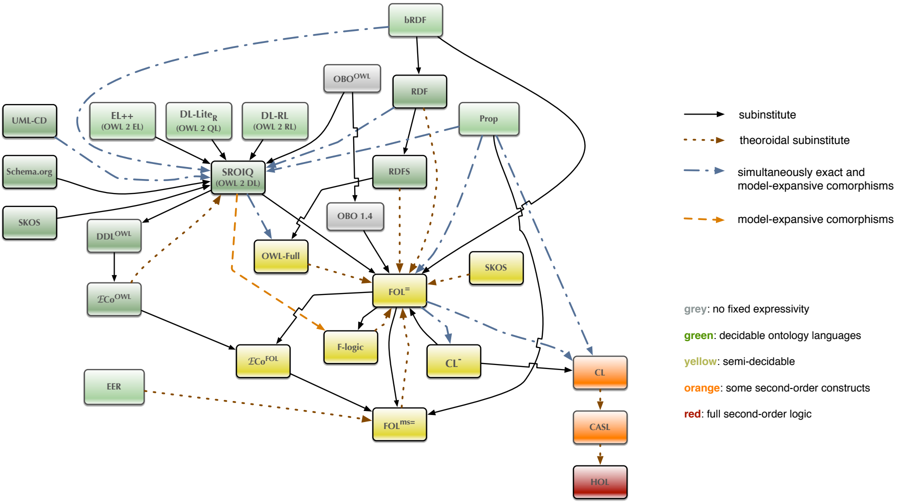
Signatures : $\Sigma \xrightarrow{\sigma} \Sigma'$

Sentences : $Sen(\Sigma) \xrightarrow{Sen(\sigma)} Sen(\Sigma')$

Satisfaction : $\models_{\Sigma}$ $\qquad$ $\models_{\Sigma'}$

Models : $Mod(\Sigma) \xleftarrow{Mod(\sigma)} Mod(\Sigma')$

DOL Semantic Foundations: Logic Translations



Tools & Ressources

Tool support: Heterogeneous Tool Set (Hets)

- available at `http://hets.eu`
- speaks DOL, propositional logic, OWL, CASL, Common Logic, QBF, modal logic, MOF, QVT, and other languages
- analysis
- computation of colimits ($\Rightarrow$ lecture 5)
- management of proof obligations
- interfaces to theorem provers, model checkers, model finders

Tool support: Ontohub web portal and repository

Ontohub is a web-based repository engine for distributed heterogeneous (multi-language) OMS

- **web-based** prototype available at `ontohub.org`
- **multi-logic** speaks the same languages as Hets
- **multiple repositories** ontologies can be organized in multiple repositories, each with its own management of editing and ownership rights,
- **Git interface** version control of ontologies is supported via interfacing the Git version control system,
- **linked-data compliant** one and the same URL is used for referencing an ontology, downloading it (for use with tools), and for user-friendly presentation in the browser.

DOL Resources

- http://dol-omg.org Central page for DOL
- http://hets.eu Analysis and Proof Tool Hets, speaking DOL
- http://ontohub.org Ontohub web platform, speaking DOL
- http://ontohub.org/dol-examples DOL examples
- http://ontoiop.org Initial standardization initiative

In particular for this course:

- https://ontohub.org/esslli-2016
  ESSLLI repository of DOL examples

Prop | FOL | OWL

Three Logics as Institutions

Following the framework of institution theory, we introduce the three logics, propositional, DL, and first-order, by outlining their

1. signatures
2. sentences
3. models
4. satisfaction relation

Propositional Logic in DOL: Signatures

The non-logical symbols are collected in a signature. In propositional logic, these are just propositional letters:

Definition (Propositional Signatures)

A propositional signature Σ is a set (of propositional letters, or propositional symbols, or propositional variables).

Propositional Logic in DOL: Sentences

A signature provides us with the basic material to form logical expressions, called formulas or sentences.

Definition (Propositional Sentences)

Given a propositional signature Σ, a propositional sentence over Σ is one produced by the following grammar

$$\phi ::= p \mid \bot \mid \top \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\phi \leftrightarrow \phi)$$

with $p \in \Sigma$. $\mathsf{Sen}(\Sigma)$ is the set of all Σ-sentences. We can omit the outermost brackets of a sentence.

Propositional Logic in DOL: Models I

Models (or Truth valuations) provide an interpretation of propositional sentences. Each propositional letter is interpreted as a truth value:

Definition (Model)

Given a propositional signature Σ, a Σ-model (or Σ-valuation) is a function $\Sigma \to \{T, F\}$. $\mathrm{Mod}(\Sigma)$ is the set of all Σ-models.

Propositional Logic in DOL: Models II

Models interpret not only the propositional letters, but all sentences.
A Σ-model M can be extended using truth tables to

$$M^{\#} : \mathrm{Sen}(\Sigma) \to \{T, F\}$$

$$M^{\#}(p) = M(p)$$
$$M^{\#}(\top) = T$$
$$M^{\#}(\bot) = F$$

(a) base cases

$M^{\#}(\phi)$	$M^{\#}(\neg\phi)$
T	F
F	T

(b) not

$M^{\#}(\phi)$	$M^{\#}(\psi)$	$M^{\#}(\phi \wedge \psi)$	$M^{\#}(\phi \vee \psi)$	$M^{\#}(\phi \to \psi)$	$M^{\#}(\phi \leftrightarrow \psi)$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

(c) and, or, implication, biimplication

Propositional Logic in DOL: Satisfaction

We now can define what it means for a sentence to be satisfied in a model:

Definition

ϕ holds in M (or M satisfies ϕ), written $M \models_\Sigma \phi$ iff

$$M^{\#}(\phi) = T$$

Prop: Example

A common formalisation of some natural language constructs is as follows:

natural language	**formalisation**
A and B	$A \wedge B$
A but B	$A \wedge B$
A or B	$A \vee B$
either A or B	$(A \vee B) \wedge \neg(A \wedge B)$
if A then B	$A \rightarrow B$
A only if B	$A \rightarrow B$
A iff B	$A \leftrightarrow B$

Theories

Common to all logics is the notion of a **theory** commonly introduced as follows. In a given logic with fixed notions of signatures, sentences, models, and satisfaction:

Definition (Theories)

A theory is a pair $T = (\Sigma, \Gamma)$ where Σ is a signature and $\Gamma \subseteq \text{Sen}(\Sigma)$. A model of a theory $T = (\Sigma, \Gamma)$ is a Σ-model M with $M \models \Gamma$. In this case T is called satisfiable.

Therefore, a propositional theory is a pair $T = (\Sigma, \Gamma)$ consisting of a set Σ of propositional variables and a set Γ of propositional formulae expressed in Σ.

Prop: Example

A scenario involving John and Maria's weekend entertainment may be written as follows in DOL (to be continued in Lecture 2):

```
logic Propositional
spec JohnMary =
  props sunny, weekend, john_tennis,
        mary_shopping, saturday
                        %% declaration of signature
. sunny /\ weekend => john_tennis %(when_tennis)%
. john_tennis => mary_shopping    %(when_shopping)%
. saturday                        %(it_is_saturday)%
. sunny                           %(it_is_sunny)%
end
```

Note: `%%` for comments and `%label%` for axiom labels.

First-order Logic in DOL: Signatures

We describe a many-sorted variant of first-order logic:

Definition

A Signature $\Sigma = (S, F, P)$ of many-sorted-FOL consists of:

- a set S of sorts, where S^* is the set of words over S
- for each $w \in S^*$,and each $s \in S$ a set $F_{w,s}$ of function symbols (here w are the argument sorts and s are the result sorts)
- for each $w \in S^*$ a set P_w of predicate symbols

First-order Logic in DOL: Terms

Definition

Given a Signature $\Sigma = (S, F, P)$ the set of ground Σ-terms is inductively defined by:

- $f_{w,s}(t_1, \ldots, t_n)$ is a term of sort s, if each t_i is a term of sort s_i $(i = 1 \ldots n, w = s_1 \ldots s_n)$ and $f \in F_{w,s}$.

In particular (for $n = 0$) this means that $w = \lambda$ (the empty word), and for $c \in F_{\lambda,s}$, c_s is a constant term of sort s.

Note: In this version of FOL, variables are not needed as terms.

First-order Logic in DOL: Sentences I

Definition

Given a signature $\Sigma = (S, F, P)$ the set of Σ-sentences is inductively defined by:

- $t_1 = t_2$ for t_1, t_2 of the same sort
- $p_w(t_1, \ldots, t_n)$ for t_i Σ-term of sort s_i,
  $(1 \leq i \leq n, w = s_1, \ldots, s_n, p \in P_w)$
- $\phi_1 \wedge \phi_2$ for ϕ_1, ϕ_2 Σ-formulae
- $\phi_1 \vee \phi_2$ for ϕ_1, ϕ_2 Σ-formulae
- $\phi_1 \rightarrow \phi_2$ for ϕ_1, ϕ_2 Σ-formulae
- $\phi_1 \leftrightarrow \phi_2$ for ϕ_1, ϕ_2 Σ-formulae
- $\neg\phi_1$ for ϕ_1 Σ-formula
- $\top$, $\bot$

First-order Logic in DOL: Sentences II

Definition (continued)

Given a signature $\Sigma = (S, F, P)$ the set of Σ-sentences is inductively defined by:

- ...
- $\forall x : s \;.\; \phi$ if $s \in S$, ϕ is a $\Sigma \uplus \{x : s\}$-sentence where $\Sigma \uplus \{x : s\}$ is Σ enriched with a new constant x of sort s
- $\exists x : s \;.\; \phi$ likewise

Note: We have no 'open formulae' in this version of FOL.

First-order Logic in DOL: Models

Definition

Given a signature $\Sigma = (S, F, P)$ a Σ-model M consists of

- a carrier set $M_s \neq \emptyset$ for each sort $s \in S$
- a function $f^m_{w,s} : M_{s_1} \times \ldots \times M_{s_n} \to M_s$ for each $f \in F_{w,s}$, $w = s_1, \ldots, s_n$.
  In particular, for a constant, this is just an element of M_s
- a relation $p^M_w \subseteq M_{s_1} \times \ldots \times M_{s_n}$ for each $p \in P_w$, $w = s_1 \ldots s_n$

First-order Logic in DOL: Evaluating Terms

Definition

A Σ-term t is evaluated in a Σ-model M as follows:

$$M(f_{w,s}(t_1, \ldots t_n)) = f_{w,s}^M(M(t_1), \ldots M(t_n))$$

First-order Logic in DOL: Satisfaction

Definition

Let $\Sigma' = \Sigma \uplus \{x : s\}$. A Σ'-model M' is called a Σ'-expansion of a Σ-model M if M' and M interpret every symbol except x in the same way.

Definition (Satisfaction of sentences)

$M \models t_1 = t_2$ iff $M(t_1) = M(t_2)$

$M \models p_w(t_1 \ldots t_n)$ iff $(M(t_1), \ldots M(t_n)) \in p_w^M$

$M \models \phi_1 \wedge \phi_2$ iff $M \models \phi_1$ and $M \models \phi_2$ etc.

$M \models \forall x : s.\phi$ iff for all Σ'-expansions M' of M, $M' \models \phi$
where $\Sigma' = \Sigma \uplus \{x : s\}$

$M \models \exists x : s.\phi$ iff there is a Σ'-expansion M' of M such that $M' \models \phi$

FOL: Example

A specification of a total order in many-sorted first-order logic, using CASL syntax:

```
logic CASL.FOL=

spec TotalOrder =
  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)%
  . forall x,y : Elem . x leq y \/ y leq x          %(dichotomy)%
end
```

Full specification at
https://ontohub.org/esslli-2016/FOL/OrderTheory.dol

OWL: Description Logic in DOL

- DOL supports the logic $\mathcal{SROIQ}$ underlying OWL 2 DL
- We focus here on the basic DL $\mathcal{ALC}$

Description Logic in DOL: Signatures

Definition

A DL-signature $\Sigma = (\mathbf{C}, \mathbf{R}, \mathbf{I})$ consists of

- a set $\mathbf{C}$ of concept names,
- a set $\mathbf{R}$ of role names,
- a set $\mathbf{I}$ of individual names,

Description Logic in DOL: Concepts

Definition

For a signature $\Sigma = (\mathbf{C}, \mathbf{R}, \mathbf{I})$ the set of $\mathcal{ALC}$-concepts[a] over Σ is defined by the following grammar:

		Manchester syntax
$C, D ::=$	A for $A \in \mathbf{C}$	**concept name**
	$\mid \top$	Thing
	$\mid \bot$	Nothing
	$\mid \neg C$	not C
	$\mid C \sqcap D$	C and D
	$\mid C \sqcup D$	C or D
	$\mid \exists R.C$ for $R \in \mathbf{R}$	R some C
	$\mid \forall R.C$ for $R \in \mathbf{R}$	R only C

[a] $\mathcal{ALC}$ stands for "attributive language with complement"

Description Logic in DOL: Sentences

Definition

The set of $\mathcal{ALC}$-Sentences over Σ (Sen(Σ)) is defined as

- $C \sqsubseteq D$, where C and D are $\mathcal{ALC}$-concepts over Σ.

  Class: C SubclassOf: D

- $a : C$, where $a \in \mathbf{I}$ and C is a $\mathcal{ALC}$-concept over Σ.

  Individual : a Types: C

- $R(a_1, a_2)$, where $R \in \mathbf{R}$ and $a_1, a_2 \in \mathbf{I}$.

  Individual : a1 Facts: R a2

Description Logic in DOL: Models I

Definition

Given $\Sigma = (\mathbf{C}, \mathbf{R}, \mathbf{I})$, a Σ-model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where

- $\Delta^{\mathcal{I}}$ is a non-empty set
- $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for each $A \in \mathbf{C}$
- $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for each $R \in \mathbf{R}$
- $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each $a \in \mathbf{I}$

Description Logic in DOL: Models II

Definition

We can extend $\cdot^{\mathcal{I}}$ to all concepts as follows:

$$
\begin{array}{lcl}
\top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} & = & \emptyset \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} & = & \{x \in \Delta^{\mathcal{I}} | \exists y \in \Delta^{\mathcal{I}}.(x,y) \in R^{\mathcal{I}}, y \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} & = & \{x \in \Delta^{\mathcal{I}} | \forall y \in \Delta^{\mathcal{I}}.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}
\end{array}
$$

Description Logic in DOL: Satisfaction

Definition (Satisfaction of sentences in a model)

$$
\begin{array}{lll}
\mathcal{I} \models C \sqsubseteq D & \text{iff} & C^{\mathcal{I}} \subseteq D^{\mathcal{I}}. \\
\mathcal{I} \models a : C & \text{iff} & a^{\mathcal{I}} \in C^{\mathcal{I}}. \\
\mathcal{I} \models R(a_1, a_2) & \text{iff} & (a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) \in R^{\mathcal{I}}.
\end{array}
$$

OWL: Example

```
logic OWL

ontology FamilyBase =
  Class: Person
  Class: Female
  Class: Woman   EquivalentTo: Person and Female
  Class: Man     EquivalentTo: Person and not Woman

  ObjectProperty: hasParent
  ObjectProperty: hasChild InverseOf: hasParent
  ObjectProperty: hasHusband

  Class: Mother  EquivalentTo: Woman and hasChild some Person
  Class: Parent  EquivalentTo: Father or Mother
  Class: Wife    EquivalentTo: Woman and hasHusband some Man
  ...
```

OWL: Example (continued)

```
  ...
  Class: Married
  Class: MarriedMother EquivalentTo: Mother and Married
         SubClassOf: Female and Person
  Individual: john Types: Father
  Individual: mary Types: Mother
  Facts: hasChild  john
end
```

Full specification at
`https://ontohub.org/esslli-2016/OWL/Family.dol`

Summary

- DOL is not a 'Lingua Franca'

Summary

- DOL is not a ‘Lingua Franca’
- DOL is a metalanguage reusing, modifying, connecting ontologies, models, and specifications (called OMS)

Summary

- DOL is not a ‘Lingua Franca’
- DOL is a metalanguage reusing, modifying, connecting ontologies, models, and specifications (called OMS)
- DOL enables a modular/structured approach to knowledge engineering

Detailed Course Overview

- **Day 1:** We just learned what DOL is about and what kind of problems it can help solve.
  Remainder of today: Get started with `Ontohub.org` and HETS.

Detailed Course Overview

- **Day 1:** We just learned what DOL is about and what kind of problems it can help solve.
  Remainder of today: Get started with `Ontohub.org` and HETS.
- **Day 2:** Basic structuring concepts for logical theories and examples in DOL

Detailed Course Overview

- **Day 1:** We just learned what DOL is about and what kind of problems it can help solve.
  Remainder of today: Get started with `Ontohub.org` and HETS.
- **Day 2:** Basic structuring concepts for logical theories and examples in DOL
- **Day 3:** General institution-theoretic semantics for DOL: logic-independence

Detailed Course Overview

- **Day 1:** We just learned what DOL is about and what kind of problems it can help solve.
  Remainder of today: Get started with `Ontohub.org` and HETS.
- **Day 2:** Basic structuring concepts for logical theories and examples in DOL
- **Day 3:** General institution-theoretic semantics for DOL: logic-independence
- **Day 4:** Working with multiple logics: heterogeneity

Detailed Course Overview

- **Day 1:** We just learned what DOL is about and what kind of problems it can help solve.
  Remainder of today: Get started with `Ontohub.org` and HETS.
- **Day 2:** Basic structuring concepts for logical theories and examples in DOL
- **Day 3:** General institution-theoretic semantics for DOL: logic-independence
- **Day 4:** Working with multiple logics: heterogeneity
- **Day 5:** Advanced applications: alignments, networks, blending

Exercise for tomorrow

- clone the ESSLLI repository on `ontohub.org`:
  `git clone git://ontohub.org/esslli-2016.git`

Exercise for tomorrow

- clone the ESSLLI repository on `ontohub.org`:
  `git clone git://ontohub.org/esslli-2016.git`
- Look at the following theories expressed in Prop / OWL
  (1) `Propositional/Penguin.dol`
  (2) `OWL/Family.dol`

Exercise for tomorrow

- clone the ESSLLI repository on `ontohub.org`:
  `git clone git://ontohub.org/esslli-2016.git`
- Look at the following theories expressed in Prop / OWL
  (1) `Propositional/Penguin.dol`
  (2) `OWL/Family.dol`
- Determine whether they are satisfiable or not

Exercise for tomorrow

- clone the ESSLLI repository on `ontohub.org`:
  `git clone git://ontohub.org/esslli-2016.git`
- Look at the following theories expressed in Prop / OWL
  (1) `Propositional/Penguin.dol`
  (2) `OWL/Family.dol`
- Determine whether they are satisfiable or not
- If they are, make them unsatisfiable, if they are not, make them satisfiable

Exercise for tomorrow

- clone the ESSLLI repository on `ontohub.org`:
  `git clone git://ontohub.org/esslli-2016.git`
- Look at the following theories expressed in Prop / OWL
  (1) `Propositional/Penguin.dol`
  (2) `OWL/Family.dol`
- Determine whether they are satisfiable or not
- If they are, make them unsatisfiable, if they are not, make them satisfiable
- Upload your results in your private Ontohub.org repository

The Distributed Ontology, Model and Specification Language (DOL) Day 2: Basic Structuring with DOL

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 1

On Day 1 we have:

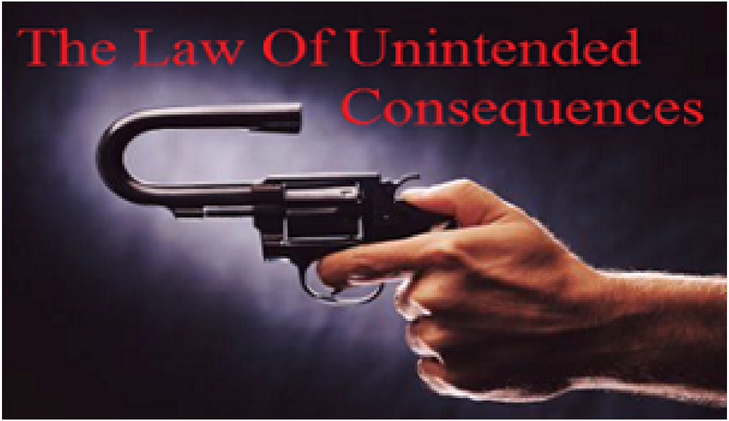
- Explored the motivation behind DOL looking at several use-cases from ontology engineering
- Introduced the basic ideas and features of DOL
- Introduced some logics we will use during the week
- Introduced the tools to be used: Ontohub and HETS

Today

We will focus today on discussing in parallel use cases for all three logics and giving DOL syntax and semantics for:

- intended consequences (competency questions)
- model finding and refutation of lemmas
- extensions and conservative extensions
- signature morphisms and the satisfaction condition
- refinements / theory interpretations

Intended Consequences



Logical Consequence in Prop, FOL and OWL

Logic deals with what follows from what.

J.A. Robinson: Logic, Form and Function.

Logical consequence = Satisfaction in a model is preserved:

$$\varphi_1, \ldots, \varphi_n \models \psi$$

All models of the premises $\varphi_1, \ldots, \varphi_n$
are models of the conclusion ψ.

Formally: $M \models \varphi_1$ and ... and $M \models \varphi_n$ together imply $M \models \psi$.

More general form:

$$\Phi \models \psi \qquad (\Phi \text{ may be infinite})$$

$M \models \varphi$ for all $\varphi \in \Phi$ implies $M \models \psi$.

Countermodels in Prop, FOL and OWL

Given a question about logical consequence over Σ-sentences,

$$\Phi \overset{?}{\models} \psi$$

a countermodel is a Σ-model M with

$$M \models \Phi \text{ and } M \not\models \psi$$

A countermodel shows that $\Phi \models \psi$ does not hold.


ATP
Do you think we should bite?
Proof
TPTP
Model

Intended Consequences in Propositional Logic

```
logic Propositional
spec JohnMary =
 props sunny, weekend, john_tennis, mary_shopping,
       saturday %% declaration of signature
 . sunny /\ weekend => john_tennis %(when_tennis)%
 . john_tennis => mary_shopping   %(when_shopping)%
 . saturday                    %(it_is_saturday)%
 . sunny                       %(it_is_sunny)%
 . mary_shopping   %(mary_goes_shopping)% %implied
end
```

Full specification at
https://ontohub.org/esslli-2016/Propositional/leisure_structured.dol

A Countermodel

```
logic Propositional
spec Countermodel =
 props sunny, weekend, john_tennis, mary_shopping,
       saturday %% declaration of signature
  . sunny
  . not weekend
  . not john_tennis
  . not mary_shopping
  . saturday
end
```

This specification has exactly one model, and hence can be seen as a syntactic description of this model.

Repaired Specification

```
logic Propositional
spec JohnMary =
 props sunny, weekend, john_tennis, mary_shopping,
       saturday %% declaration of signature
 . sunny /\ weekend => john_tennis %(when_tennis)%
 . john_tennis => mary_shopping   %(when_shopping)%
 . saturday                 %(it_is_saturday)%
 . sunny                    %(it_is_sunny)%
 . saturday => weekend %(sat_weekend)%
 . mary_shopping   %(mary_goes_shopping)% %implied
end
```

Intended Consequences in FOL

```
logic CASL.FOL=
spec BooleanAlgebra =
  sort Elem
  ops 0,1 : Elem;
      __ cap __ : Elem * Elem -> Elem, assoc, comm, unit 1;
      __ cup __ : Elem * Elem -> Elem, assoc, comm, unit 0;
  forall x,y,z:Elem
  . x cap (x cup y) = x     %(absorption_def1)%
  . x cup (x cap y) = x     %(absorption_def2)%
  . x cap 0 = 0             %(zeroAndCap)%
  . x cup 1 = 1             %(oneAndCup)%
  . x cap (y cup z) = (x cap y) cup (x cap z)
                            %(distr1_BooleanAlgebra)%
  . x cup (y cap z) = (x cup y) cap (x cup z)
                            %(distr2_BooleanAlgebra)%
  . exists x' : Elem . x cup x' = 1 /\ x cap x' = 0
                            %(inverse_BooleanAlgebra)%
  . x cup x = x             %(idem_cup)% %implied
  . x cap x = x             %(idem_cap)% %implied
end
```

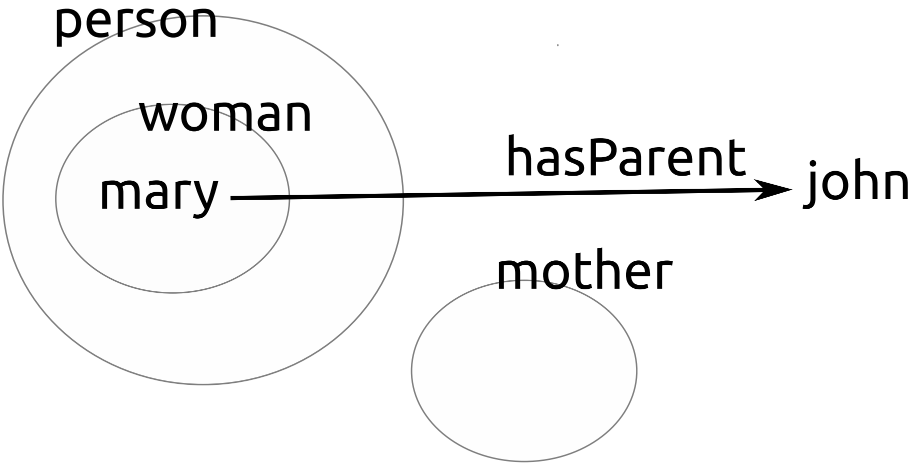
https://ontohub.org/esslli-2016/FOL/OrderTheory_structured.dol

Intended Consequences in OWL

```
logic OWL
ontology Family1 =
  Class: Person
  Class: Woman SubClassOf: Person
  ObjectProperty: hasChild
  Class: Mother
          EquivalentTo: Woman and hasChild some Person
  Individual: mary Types: Woman Facts: hasChild  john
  Individual: john
  Individual: mary
        Types: Annotations: Implied "true"^^xsd:boolean
               Mother
end
```

https://ontohub.org/esslli-2016/OWL/Family_structured.dol

A Countermodel



Repaired Ontology

```
logic OWL
ontology Family2 =
  Class: Person
  Class: Woman SubClassOf: Person
  ObjectProperty: hasChild
  Class: Mother
         EquivalentTo: Woman and hasChild some Person
  Individual: mary Types: Woman Facts: hasChild  john
  Individual: john Types: Person
  Individual: mary
        Types: Annotations: Implied "true"^^xsd:boolean
               Mother
end
```

Extensions

Structuring Using Extensions

```
logic Propositional
spec JohnMary_TBox = %% general rules
  props sunny, weekend, john_tennis, mary_shopping,
        saturday   %% declaration of signature
  . sunny /\ weekend => john_tennis %(when_tennis)%
  . john_tennis => mary_shopping   %(when_shopping)%
  . saturday => weekend  %(sat_weekend)%
end
spec JohnMary_ABox = %% specific facts
  JohnMary_TBox then
  . saturday                 %(it_is_saturday)%
  . sunny                    %(it_is_sunny)%
  . mary_shopping    %(mary_goes_shopping)% %implied
end
```

Implied Extensions in Prop

```
logic Propositional
spec JohnMary_variant =
  props sunny, weekend, john_tennis, mary_shopping,
        saturday   %% declaration of signature
  . sunny /\ weekend => john_tennis %(when_tennis)%
  . john_tennis => mary_shopping   %(when_shopping)%
  . saturday => weekend  %(sat_weekend)%
then
  . saturday                 %(it_is_saturday)%
  . sunny                    %(it_is_sunny)%
then %implies
  . mary_shopping     %(mary_goes_shopping)%
end
```

Implied Extensions in OWL

```
ontology Family1 =
  Class: Person
  Class: Woman SubClassOf: Person
  ObjectProperty: hasChild
  Class: Mother
     EquivalentTo: Woman and hasChild some Person
  Individual: john Types: Person
  Individual: mary Types: Woman Facts: hasChild john
then %implies
  Individual: mary Types: Mother
end
```

Conservative Extensions in Prop

```
logic Propositional
spec Animals =
  props bird, penguin, living
  . penguin => bird
  . bird => living
then %cons
  prop animal
  . bird => animal
  . animal => living
end
```

In the extension, no "new" facts about the "old" signature follow.

A Non-Conservative Extension

```
spec Animals =
  props bird, penguin, living
  . penguin => bird
then %% not a conservative extension
  prop animal
  . bird => animal
  . animal => living
end
```

In the extension, “new” facts about the “old” signature follow, namely

```
  . bird => living
```

A Conservative Extension in FOL

```
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 TotalOrder = PartialOrder then
  . forall x,y:Elem. x leq y \/ y leq x        %(dichotomy)%
then %cons
  pred __ < __ : Elem * Elem
  . forall x,y:Elem. x < y <=> (x leq y /\ not x = y)
                                                  %(<-def)%
end
```

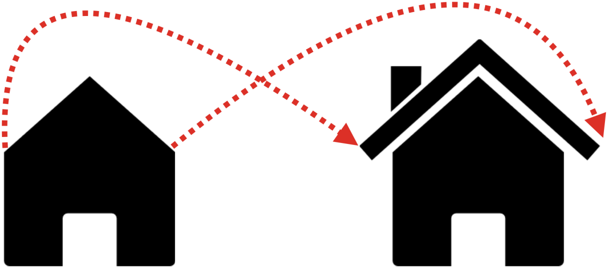
A Conservative Extension in OWL

```
logic OWL
ontology Animals1 =
  Class: LivingBeing
  Class: Bird SubClassOf: LivingBeing
  Class: Penguin SubClassOf: Bird
then %cons
  Class: Animal SubClassOf: LivingBeing
  Class: Bird SubClassOf: Animal
end
```

A Nonconservative Extension in OWL

```
logic OWL
ontology Animals2 =
  Class: LivingBeing
  Class: Bird
  Class: Penguin SubClassOf: Bird
then %% not a conservative extension
  Class: Animal SubClassOf: LivingBeing
  Class: Bird SubClassOf: Animal
end
```

Signature Morphisms and the Satisfaction Condition



Signature morphisms in propositional logic

Definition

Given two propositional signatures Σ_1, Σ_2 a signature morphism is a function $\sigma : \Sigma_1 \to \Sigma_2$. (Note that signatures are sets.)

Definition

A signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ induces a sentence translation $\mathsf{Sen}(\Sigma_1) \to \mathsf{Sen}(\Sigma_2)$, by abuse of notation also denoted by σ, defined inductively by

- $\sigma(p) = \sigma(p)$ (the two σs are different. . .)
- $\sigma(\bot) = \bot$
- $\sigma(\top) = \top$
- $\sigma(\phi_1 \wedge \phi_2) = \sigma(\phi_1) \wedge \sigma(\phi_2)$
- etc.

Model reduction in propositional logic

Definition

A signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ induces a model reduction function

$$\_|_\sigma : \mathrm{Mod}(\Sigma_2) \to \mathrm{Mod}(\Sigma_1).$$

Given $M \in \mathrm{Mod}(\Sigma_2)$ i.e. $M : \Sigma_2 \to \{T, F\}$,
then $M|_\sigma \in \mathrm{Mod}(\Sigma_1)$ is defined as

$$M|_\sigma(p) := M(\sigma(p))$$

for all $p \in \Sigma_1$, i.e.

$$M|_\sigma = M \circ \sigma$$

If $M'|_\sigma = M$, then M' is called a σ-expansion of M.

Satisfaction condition in propositional logic

Theorem (Satisfaction condition)

Given a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$, $M_2 \in Mod(\Sigma_2)$ *and* $\phi_1 \in Sen(\Sigma_1)$, *then:*

$$M_2 \models_{\Sigma_2} \sigma(\phi_1) \text{ iff } M_2|_\sigma \models_{\Sigma_1} \phi_1$$

(*"truth is invariant under change of notation."*)

Proof.

By induction on ϕ_1. □

Signature Morphisms in FOL

Definition

Given signatures $\Sigma = (S, F, P), \Sigma' = (S', F', P')$ a signature morphism $\sigma : \Sigma \to \Sigma'$ consists of

- a map $\sigma^S : S \to S'$
- a map $\sigma^F_{w,s} : F_{w,s} \to F'_{\sigma^S(w),\sigma^S(s)}$ for each $w \in S^*$ and each $s \in S$
- a map $\sigma^P_w : P_w \to P'_{\sigma^S(w)}$ for each $w \in S^*$

Model Reduction in FOL

Definition

Given a signature morphism $\sigma : \Sigma \to \Sigma'$ and a Σ'-model M', define $M = M'|_\sigma$ as

- $M_s = M'_{\sigma^S(s)}$
- $f^M_{w,s} = \sigma^F_{w,s}(f)^{M'}_{\sigma^S(w),\sigma^S(s)}$
- $p^M_{w,s} = \sigma^P_w(p)^{M'}_{\sigma^S(w)}$

Sentence Translation in FOL

Definition

Given a signature morphism $\sigma : \Sigma \to \Sigma'$ and $\phi \in \mathrm{Sen}(\Sigma)$ the translation $\sigma(\phi)$ is defined inductively by:

$$
\begin{aligned}
\sigma(f_{w,s}(t_1 \dots t_n)) &= \sigma^F_{w,s}(f_{\sigma(w),\sigma(s))}(\sigma(t_1) \dots \sigma(t_n)) \\
\sigma(t_1 = t_2) &= \sigma(t_1) = \sigma(t_2) \\
\sigma(p_w(t_1 \dots t_n)) &= \sigma^P_w(p)_{\sigma^S(w)}(\sigma(t_1) \dots \sigma(t_n)) \\
\sigma(\phi_1 \wedge \phi_2) &= \sigma(\phi_1) \wedge \sigma(\phi_2) \qquad \text{etc.} \\
\sigma(\forall x : s.\phi) &= \forall x : \sigma^S(s).(\sigma \uplus x)(\phi) \\
\sigma(\exists x : s.\phi) &= \exists x : \sigma^S(s).(\sigma \uplus x)(\phi)
\end{aligned}
$$

where $(\sigma \uplus x) : \Sigma \uplus \{x : s\} \to \Sigma' \uplus \{x : \sigma(s)\}$ acts like σ on Σ and maps $x : s$ to $x : \sigma(s)$.

First-order Logic in DOL: Satisfaction Revisited

Definition (Satisfaction of sentences)

$M \models t_1 = t_2$ iff $M(t_1) = M(t_2)$

$M \models p_w(t_1 \ldots t_n)$ iff $(M(t_1), \ldots M(t_n)) \in p_w^M$

$M \models \phi_1 \wedge \phi_2$ iff $M \models \phi_1$ and $M \models \phi_2$

$M \models \forall x : s.\phi$ iff for all ι-expansions M' of M, $M' \models \phi$
where $\iota : \Sigma \hookrightarrow \Sigma \uplus \{x : s\}$ is the inclusion.

$M \models \exists x : s.\phi$ iff there is a ι-expansion M' of M such that $M' \models \phi$

Satisfaction Condition in FOL

Theorem (satisfaction condition)

For a signature morphism $\sigma : \Sigma \to \Sigma', \phi \in Sen(\Sigma), M' \in Mod(\Sigma')$:

$$M'|_\sigma \models \phi \textit{ iff } M' \models \sigma(\phi)$$

Proof.

For terms, prove $M'|_\sigma(t) = M'(\sigma(t))$. Then use induction on ϕ. For quantifiers, use a bijective correspondence between ι-expansions M_1 of $M'|_\sigma$ and ι'-expansions M'_1 of M'.

$$\begin{array}{ccccc} M'|_\sigma & & \Sigma \xrightarrow{\ \sigma\ } \Sigma' & & M' \\ & & \iota \downarrow \quad\quad \downarrow \iota' & & \\ M_1 & \Sigma \uplus \{x : s\} = \Sigma_1 & \xrightarrow{\ \sigma \uplus x\ } & \Sigma'_1 = \Sigma' \uplus \{x : \sigma(s)\} & M'_1 \end{array}$$

Signature Morphisms in OWL

Definition

Given two DL signatures $\Sigma_1 = (\mathbf{C}_1, \mathbf{R}_1, \mathbf{I}_1)$ and $\Sigma_2 = (\mathbf{C}_2, \mathbf{R}_2, \mathbf{I}_2)$ a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ consists of three functions

- $\sigma^C : \mathbf{C}_1 \to \mathbf{C}_2$,
- $\sigma^R : \mathbf{R}_1 \to \mathbf{R}_2$,
- $\sigma^I : \mathbf{I}_1 \to \mathbf{I}_2$.

Sentence Translation in OWL

Definition

Given a signature morphism $\sigma : \Sigma_1 \rightarrow \Sigma_2$ and a Σ_1-sentence ϕ, the translation $\sigma(\phi)$ is defined by inductively replacing the symbols in ϕ along σ.

Model Reduction in OWL

Definition

Given a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ and a Σ_2-model $\mathcal{I}_2$, the σ-reduct of $\mathcal{I}_2$ along σ is the Σ_1-model $\mathcal{I}_1 = \mathcal{I}_2|_\sigma$ defined by

- $\Delta^{\mathcal{I}_1} = \Delta^{\mathcal{I}_2}$
- $A^{\mathcal{I}_1} = \sigma^C(A)^{\mathcal{I}_2}, \quad \text{for } A \in \mathbf{C}_1$
- $R^{\mathcal{I}_1} = \sigma^R(R)^{\mathcal{I}_2}, \quad \text{for } R \in \mathbf{R}_1$
- $a^{\mathcal{I}_1} = \sigma^I(a)^{\mathcal{I}_2}, \quad \text{for } a \in \mathbf{I}_1$

Satisfaction Condition in OWL

Theorem (satisfaction condition)

Given $\sigma : \Sigma_1 \to \Sigma_2$, $\phi_1 \in Sen(\Sigma_1)$ *and* $\mathcal{I}_2 \in Mod(\Sigma_2)$,

$$\mathcal{I}_2|_\sigma \models \phi_1 \quad \textit{iff} \quad \mathcal{I}_2 \models \sigma(\phi_1)$$

Proof.

Let $\mathcal{I}_1 = \mathcal{I}_2|_\sigma$. Note that $\mathcal{I}_1$ and $\mathcal{I}_2$ share the universe: $\Delta^{\mathcal{I}_1} = \Delta^{\mathcal{I}_2}$.
First prove by induction over concepts C that

$$C^{\mathcal{I}_1} = \sigma(C)^{\mathcal{I}_2}.$$

Then the satisfaction condition follows easily. □

Theory Morphisms in Prop, FOL, OWL

Definition

A theory morphism $\sigma : (\Sigma_1, \Gamma_1) \to (\Sigma_2, \Gamma_2)$ is a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ such that

$$\text{for } M \in \mathrm{Mod}(\Sigma_2, \Gamma_2), \text{ we have } M|_\sigma \in \mathrm{Mod}(\Sigma_1, \Gamma_1)$$

Extensions are theory morphisms:

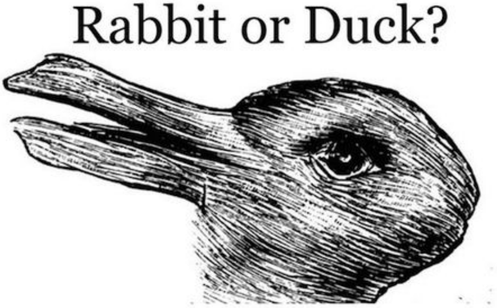
$$(\Sigma, \Gamma) \textbf{ then } (\Delta_\Sigma, \Delta_\Gamma)$$

leads to the theory morphism

$$(\Sigma, \Gamma) \xrightarrow{\iota} (\Sigma \cup \Delta_\Sigma, \iota(\Gamma) \cup \Delta_\Gamma)$$

Proof: $M \models \iota(\Gamma) \cup \Delta_\Gamma$ implies $M|_\iota \models \Gamma$ by the satisfaction condition.

Interpretations



Interpretations (views, refinements)

- **interpretation** *name* : O_1 **to** $O_2 = \sigma$
- σ is a signature morphism (if omitted, assumed to be identity)
- expresses that σ is a theory morphism $O_1 \rightarrow O_2$

```
logic CASL.FOL=
spec RichBooleanAlgebra =
  BooleanAlgebra
then %def
  pred __ <= __ : Elem * Elem;
  forall x,y:Elem
  . x <= y <=> x cap y = x %(leq_def)%
end
interpretation order_in_BA :
  PartialOrder to RichBooleanAlgebra
end
```

Recall Family Ontology

```
logic OWL
ontology Family2 =
  Class: Person
  Class: Woman SubClassOf: Person
  ObjectProperty: hasChild
  Class: Mother
         EquivalentTo: Woman and hasChild some Person
  Individual: mary Types: Woman Facts: hasChild  john
  Individual: john Types: Person
  Individual: mary
        Types: Annotations: Implied "true"^^xsd:boolean
               Mother
end
```

Interpretation in OWL

```
logic OWL
ontology Family_alt =
  Class: Human
  Class: Female
  Class: Woman EquivalentTo: Human and Female
  ObjectProperty: hasChild
  Class: Mother
         EquivalentTo: Female and hasChild some Human
end

interpretation i : Family_alt to Family2 =
  Human |-> Person, Female |-> Woman
end
```

Criterion for Theory Morphisms in Prop, FOL, OWL

Theorem

A signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ *is a theory morphism* $\sigma : (\Sigma_1, \Gamma_1) \to (\Sigma_2, \Gamma_2)$ *iff*

$$\Gamma_2 \models_{\Sigma_2} \sigma(\Gamma_1)$$

Proof.

By the satisfaction condition. □

Implied extensions (in Prop, FOL, OWL)

The extension must not introduce new signature symbols:

$$(\Sigma, \Gamma)\ \textbf{then}\ (\emptyset, \Delta_\Gamma)$$

This leads to the theory morphism

$$(\Sigma, \Gamma) \xrightarrow{\iota} (\Sigma, \Gamma \cup \Delta_\Gamma)$$

The implied extension is well-formed if

$$\Gamma \models_\Sigma \Delta_\Gamma$$

That is, implied extensions are about logical consequence.

Conservative Extensions (in Prop, FOL, OWL)

Definition

A theory morphism $\sigma : T_1 \rightarrow T_2$ is consequence-theoretically conservative (ccons), if for each $\phi_1 \in \mathsf{Sen}(\Sigma_1)$

$$T_2 \models \sigma(\phi_1) \text{ implies } T_1 \models \phi_1.$$

(no "new" facts over the "old" signature)

Definition

A theory morphism $\sigma : T_1 \rightarrow T_2$ is model-theoretically conservative (mcons), if for each $M_1 \in \mathsf{Mod}(T_1)$, there is a σ-expansion

$$M_2 \in \mathsf{Mod}(T_2) \text{ with } (M_2)|_\sigma = M_1$$

A General Theorem

Theorem

In propositional logic, FOL and OWL, if $\sigma : T_1 \to T_2$ is mcons, then it is also ccons.

Proof.

Assume that $\sigma : T_1 \to T_2$ is mcons.
Let ϕ_1 be a formula, such that $T_2 \models_{\Sigma_2} \sigma(\phi_1)$.
Let M_1 be a model $M_1 \in Mod(T_1)$. By assumption there is a model $M_2 \in Mod(T_2)$ with $M_2|_\sigma = M_1$. Since $T_2 \models_{\Sigma_2} \sigma(\phi_1)$, we have $M_2 \models \sigma(\phi_1)$. By the satisfaction condition $M_2|_\sigma \models_{\Sigma_1} \phi_1$. Hence $M_1 \models \phi_1$. Altogether $T_1 \models_{\Sigma_1} \phi_1$. □

Some prerequisites

Theorem (Compactness theorem for propositional logic)

If $\Gamma \models_\Sigma \phi$*, then* $\Gamma' \models_\Sigma \phi$ *for some finite* $\Gamma' \subseteq \Gamma$

Proof.

Logical consequence $\models_\Sigma$ *can be captured by provability* $\vdash_\Sigma$*. Proofs are finite.* □

Definition

Given a model $M \in \mathsf{Mod}(\Sigma)$, its theory $Th(M)$ is defined by

$$Th(M) = \{\varphi \in \mathsf{Sen}(\Sigma) \mid M \models_\Sigma \varphi\}$$

In Prop, the converse holds

Theorem

In propositional logic, if $\sigma : T_1 \to T_2$ is ccons, then it is also mcons.

Proof.

Assume that $\sigma : T_1 \to T_2$ *is ccons. Let* M_1 *be a model* $M_1 \in Mod(T_1)$. *Assume that* M_1 *has no* σ*-expansion to a* T_2*-model. This means that* $T_2 \cup \sigma(Th(M_1)) \models \bot$. *Hence by compactness we have* $T_2 \cup \sigma(\Gamma) \models \bot$ *for a finite* $\Gamma \subseteq Th(M_1)$. *Let* $\Gamma = \{\phi_1, \ldots, \phi_n\}$. *Thus* $T_2 \cup \sigma(\{\phi_1, \ldots, \phi_n\}) \models \bot$ *and hence* $T_2 \models \sigma(\phi_1) \wedge \ldots \wedge \sigma(\phi_n) \to \bot$. *This means* $T_2 \models \sigma(\phi_1 \wedge \ldots \wedge \phi_n \to \bot)$. *By assumption* $T_1 \models \phi_1 \wedge \ldots \wedge \phi_n \to \bot$. *Since* $M_1 \in Mod(T_1)$ *and* $M_1 \models \phi_i$ $(1 \leq i \leq n)$, *also* $M_1 \models \bot$. *Contradiction!* □

A Counterexample in ALC (ccons, not mcons)

```
logic OWL.ALC
ontology Service =
  ObjectProperty: provider
  ObjectProperty: input
  ObjectProperty: output
  Class: Webservice SubClassOf: provider some Thing
          and input some Thing and output some Thing
then %ccons
  Class: Array
  Class: Integer DisjointWith: Array
  Class: Webservice SubClassOf: input some Integer
         and input some Array
end
```

In OWL.SROIQ, this is not even ccons!

A Counterexample in FOL (ccons, not mcons)

```
logic CASL.FOL=
spec Weak_Nat =
  sort Nat ops 0:Nat succ: Nat -> Nat pred __<__ : Nat*Nat
  forall x,y,z : Nat
  . x = 0 \/ exists u:Nat . succ(u) = x
  . x < succ(y) <=> (x<y \/ x = y)
  . not (x < 0)
  . x < y => not (y < x)
  . (x < y /\ y < z) => (x < z)
  . x < y \/ x = y \/ y < x
then %ccons
  op __ + __ : Nat * Nat -> Nat
  forall x,y : Nat
  . 0 + y = y
  . succ(x) + y = succ(x + y) %(+succ)%
  . y < succ(x) + y           %(succ_great)%   end
```

Definitional Extensions (in Prop, FOL, OWL)

Definition

A theory morphism $\sigma : T_1 \to T_2$ is definitional, if for each $M_1 \in \mathrm{Mod}(T_1)$, there is a unique σ-expansion

$$M_2 \in \mathrm{Mod}(T_2) \text{ with } (M_2)|_\sigma = M_1$$

```
logic Propositional
spec Person =
  props person, male, female
then %def
  props man, woman
  . man <=> person /\ male
  . woman <=> person /\ female
end
```

Definitional Extensions: Example in OWL

```
logic OWL
ontology Person =
  Class: Person
  Class: Female
then %def
  Class: Woman  EquivalentTo: Person and Female
end
```

Summary of DOL Syntax for Extensions

- O_1 **then %mcons** O_2, O_1 **then %mcons** O_2: model-conservative extension
  - each O_1-model has an expansion to O_1 **then** O_2
- O_1 **then %ccons** O_2: consequence-conservative extension
  - O_1 **then** $O_2 \models \varphi$ implies $O_1 \models \varphi$, for φ in the language of O_1
- O_1 **then %def** O_2: definitional extension
  - each O_1-model has a unique expansion to O_1 **then** O_2
- O_1 **then %implies** O_2: implied extension
  - like %mcons, but O_2 must not extend the signature

Scaling it to the Web

- OMS can be referenced directly by their URL (or IRI)

```
<http://owl.cs.manchester.ac.uk/co-ode-files/ontologies/
pizza.owl>
```

- Prefixing may be used for abbreviation

```
%prefix( co-ode:
  <http://owl.cs.manchester.ac.uk/co-ode-files/ontologies/>
       )%
co-ode:pizza.owl
```

Exercise for tomorrow

- if you not have done so already, clone the ESSLLI repository on `ontohub.org`:
  `git clone git://ontohub.org/esslli-2016.git`

Exercise for tomorrow

- if you not have done so already, clone the ESSLLI repository on `ontohub.org`:
  `git clone git://ontohub.org/esslli-2016.git`
- Look at the theories

Exercise for tomorrow

- if you not have done so already, clone the ESSLLI repository on `ontohub.org`:
  `git clone git://ontohub.org/esslli-2016.git`
- Look at the theories
- (Dis)prove theorems (both with Hets and on Ontohub.org)

Exercise for tomorrow

- if you not have done so already, clone the ESSLLI repository on `ontohub.org`:
  `git clone git://ontohub.org/esslli-2016.git`
- Look at the theories
- (Dis)prove theorems (both with Hets and on Ontohub.org)
- Write some theory on your own, add intended consequences and prove them

The Distributed Ontology, Model and Specification Language (DOL)
Day 3: Structured OMS

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 2

On Day 2 we have looked at:

- intended consequences (competency questions)
- model finding and refutation of lemmas
- extensions and conservative extensions
- refinements / theory interpretations

Today

We will focus today on structured OMS:

- Assembling OMS from pieces:
  Basic OMS, union, translation
- Making a large OMS smaller:
  module extraction, approximation, reduction, filtering
- Non-monotonic reasoning through employing
  a closed-world assumption:
  minimization, maximization, freeness, cofreeness

Assembling OMS from Pieces

Unions

O_1 **and** O_2: union of two stand-alone OMS

- Signatures (and axioms) are united
- model classes are intersected
- difference to extensions: there, O_2 needs to be basic

```
logic CASL.FOL=
spec Magma =
  sort Elem;  ops 0:Elem; __+__:Elem*Elem->Elem  end
spec CommutativeMagma = Magma then
  forall x,y:Elem . x+y=y+x                      end
spec Monoid = Magma then
  forall x,y,z:Elem . x+0=x
                    . x+(y+z) = (x+y)+z          end
spec CommutativeMonoid =
  CommutativeMagma and Monoid                    end
```

Competency Questions Revisited

Competency Questions – Simplified Summary

- Let O be an ontology
- Capture requirements for O as pairs of scenarios and competency questions
- For each scenario competency question pair S, Q:
  - Formalize S, resulting in theory Γ
  - Formalize Q, resulting in formula φ
  - Check with theorem prover whether $O \cup \Gamma \models \varphi$
- When all proofs are successful, your ontology meets the requirements.

Competency Questions Revisited

- CQ most successful idea for ontology evaluation
- Technically, CQ = proof obligations
- Language for expressing proof obligations?
- Ad hoc handling of CQs

We asked:

- How do we keep track of scenarios and competency questions in a systematic way?

Answer: The DOL constructs of `and` (union) and `%implies`

Competency Questions Workflow

1. The use cases for the ontology are captured in form of scenarios. Each scenario describes a possible state of the world and raises a set of competency questions. The answers to these competency questions should follow logically from the scenario – provided the knowledge that is supposed to be represented in the ontology.
2. A scenario and its competency questions are formalized or an existing formalization is refined.
3. The ontology is (further) developed.
4. An automatic theorem prover is used to check whether the competency questions logically follow from the scenario and the ontology.
5. Steps (2-4) are repeated until all competency questions can be proven from the combination of the ontology and their respective scenarios.

CQ Example: Family Relations

Ontohub enables the representation and execution of competency questions with the help of DOL files.

The use case is to enable semantically enhanced searches for a database, which contains names of people, their gender, and information about parenthood. Assuming the database contains the following information:

- *Amy is female and a parent of Berta and Chris.*
- *Berta is female.*
- *Chris is male and a parent of Dora.*
- *Dora is female.*

CQ Example: Family Relations (continued)

In this case the system should be able to answer the following questions:

- *Is Chris a father? (expected: yes)*
- *Is Dora a child of Chris (expected: yes)*
- *Is Chris female? (expected: no)*
- *Is Amy older than Dora? (expected: yes)*
- *Is Berta older than Chris (expected: unknown)*

CQ Example: Input Ontology

The ontology just discussed could be represented as follows.

```
logic OWL

ontology genealogy =
  Class: Male
  Class: Female

  ObjectProperty: parent_of
  Characteristics:  Irreflexive,  Asymmetric
  SubPropertyOf: older_than

  Class: Father
  EquivalentTo: parent_of some owl:Thing and Male

  ObjectProperty: child_of
  InverseOf: parent_of

  DisjointClasses: Male, Female

  ObjectProperty: older_than
  Characteristics:  Transitive
end
```

CQ Example: Scenario Formalisation

```
ontology  scenario =
  Class: Male
  Class: Female
  ObjectProperty: parent_of

  Individual: Amy
  Types: Female
  Facts: parent_of  Berta
  Facts: parent_of  Chris

  Individual: Berta
  Types: Female

  Individual: Chris
  Types: Male
  Facts: parent_of Dora

  Individual: Dora
  Types: Female
end
```

CQ Example: Competency Questions Formalisation

```
ontology CCbase = genealogy  and scenario
%% Is Chris a father? (expected: yes)
ontology CC1 =  CCbase then %implies
  { Individual: Chris
    Types: Father }
%% Is Dora a child of Chris (expected: yes)
ontology CC2 =  CCbase then %implies
  { Individual: Dora
    Facts: child_of Chris }
%% Is Chris female? (expected: no)
%% reformulated: Is Chris not female? (expected: yes)
ontology CC3 =  CCbase then %implies
  { Individual: Chris
    Types: not Female }
%% Is Amy older than Dora? (expected: yes)
ontology CC4 =  CCbase then %implies
  { Individual: Amy
    Facts: older_than Dora }
%% Is Berta older than Chris (expected: unknown)
ontology CC5 =  CCbase then %satisfiable
  { Individual: Berta
    Facts: older_than Chris }
```

CQ approach applied to machine diagnosis

Suppose the engine of a car does not perform properly. We want to decide whether we should

- repair the engine,
- replace the engine, or
- replace auxiliary equipment.

Some Rules for Machine Diagnosis

The following facts relate symptoms to diagnoses:

(i) If the engine overheats and the ignition is correct, then the radiator is clogged.
(ii) If the engine emits a pinging sound under load and the ignition timing is correct, then the cylinders have carbon deposits.
(iii) If power output is low and the ignition timing is correct, then the piston rings are worn, or the carburetor is defective, or the air filter is clogged.
(iv) If the exhaust fumes are black, then the carburetor is defective, or the air filter is clogged.
(v) If the exhaust fumes are blue, then the piston rings are worn, or the valve seals are worn.
(vi) The compression is low if and only if the piston rings are worn.

Some Rules for Machine Diagnosis

The following facts relate diagnoses to repair decisions:

(i) If the piston rings are worn, then the engine should be replaced.

(ii) If carbon deposits are present in the cylinders or the carburetor is defective or valve seals are worn, then the engine should be repaired.

(iii) If the air filter or radiator is clogged, then that equipment should be replaced.

Machine Diagnosis: Input Specification

```
logic Propositional

%% possible symptoms of an engine that is malfunctioning
spec EngineSymptoms =
  props black_exhaust, blue_exhaust, low_power, overheat,
        ping, incorrect_timing, low_compression
end

%% diagnosis derived from symptoms
spec EngineDiagnosis = EngineSymptoms then %cons
  props carbon_deposits, clogged_filter, clogged_radiator,
defective_carburetor, worn_rings, worn_seals
   . overheat /\ not incorrect_timing => clogged_radiator  %(diagnosis1)%
   . ping /\ not incorrect_timing => carbon_deposits       %(diagnosis2)%
   . low_power /\ not incorrect_timing =>
                worn_rings \/ defective_carburetor \/ clogged_filter
                          %(diagnosis3)%
   . black_exhaust => defective_carburetor \/ clogged_filter %(diagnosis4)%
   . blue_exhaust => worn_rings \/ worn_seals                %(diagnosis5)%
   . low_compression <=> worn_rings                          %(diagnosis6)%
end
```

Machine Diagnosis: Input Specification (cont'd)

```
%% needed repair, derived from diagnosis
spec EngineRepair = EngineDiagnosis
then %cons
  props replace_auxiliary,
repair_engine,
replace_engine
  . worn_rings => replace_engine                %(rule_replace_engine)%
  . carbon_deposits \/ defective_carburetor \/ worn_seals => repair_engine
                                                %(rule_repair_engine)%
  . clogged_filter \/ clogged_radiator => replace_auxiliary
                                                %(rule_replace_auxiliary)%
end
```

Machine Diagnosis: Scenario Formalisation

Suppose the car owner complains that the engine overheats. Due to a recent engine check, it is known that the ignition timing is correct. What should be done to eliminate the problem?

```
spec MyObservedSymptoms =
  EngineSymptoms
then
  . overheat                %(symptom_overheat)%
  . not incorrect_timing    %(symptom_not_incorrect_timing)%
end
```

Diagnosis Question Formalisation

```
spec MyRepair =
  EngineRepair and MyObservedSymptoms
end

spec Repair =
  prop repair
  . repair
end

interpretation repair1 : Repair to MyRepair = %cons
  repair |-> replace_engine end
interpretation repair2 : Repair to MyRepair = %cons
  repair |-> repair_engine end
interpretation repair3 : Repair to MyRepair = %cons
  repair |-> replace_auxiliary end
%% only repair3 is a valid interpretation. That is, 'replace_auxiliary'
%% is the required action
```

Translations

A translation O **with** σ renames O along σ

- σ is a signature morphism
- in practice, σ is a symbol map, from which one can compute a signature morphism

```
ontology BankOntology =
  Class: Bank  Class: Account ...     end
ontology RiverOntology =
  Class: River  Class: Bank ...       end
ontology Combined =
  BankOntology with Bank |-> FinancialBank
 and
  RiverOntology with Bank |-> RiverBank
    %% necessary disambiguation when uniting OMS
end
```

Making large OMS smaller

Making a large OMS smaller

General problem:

you have an OMS over a large signature Σ *and want to make it smaller. Say, it should be restricted to* $\Sigma' \subseteq \Sigma$.

DOL provides four options:

- Module extraction
- Approximation
- Reduction
- Filtering

We will discuss these options for two examples:

- the medical ontology SNOMED
- the specification of groups

Module Extraction applied to SNOMED

Question: What does SNOMED say about hearts and heart attacks?

Answer 1:

```
SNOMED extract Heart, HeartAttack
```

extract:

- SNOMED module (sub-ontology of SNOMED)
- capturing the same facts about hearts and heart attacks as SNOMED itself (SNOMED is a conservative extension of the module)
- signature of the module may contain more than heart and heart attack

Dual operation: **remove** (lists the symbols to remove)

Approximation applied to SNOMED

Question: What does SNOMED say about hearts and heart attacks?

Answer 2:

```
SNOMED keep Heart, HeartAttack
```

keep:

- captures all logical consequences involving `Heart(Attack)`
- not necessarily a sub-OMS
- may involve new axioms in order to capture the SNOMED facts about hearts and heart attacks
- resulting OMS features exactly the two specified entities, heart and heart attack
- finite axiomatization may be hard to compute, if it exists at all

Dual operation: **`forget`** (lists the symbols to remove)

Reduction applied to SNOMED

Question: What does SNOMED say about hearts and heart attacks?

Answer 3:

`SNOMED reveal Heart, HeartAttack`

reveal:

- essentially keeps the whole of SNOMED
- provides some export interface consisting of heart and heart attack only
- while symbols are hidden, the semantic effect of sentences (also those involving these symbols) is kept
- useful when interfacing SNOMED with other ontologies, e.g. in an interpretation.

Dual operation: **`hide`** (lists the symbols to remove)

Filtering applied to SNOMED

Question: What does SNOMED say about hearts and heart attacks?

Answer 4:

```
SNOMED select Heart, HeartAttack
```

select:

- simply removes all SNOMED axioms that involve other symbols then heart and heart attack
- can be computed easily
- might lead to poor ontology, capturing only a small fraction and only the basic facts of SNOMED's knowledge about hearts and heart attacks.

Dual operation: **`reject`** (lists the symbols to remove)

Module Extraction applied to Groups (1)

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
forall x,y,z:elem . x+0=x
                  . x+(y+z) = (x+y)+z
                  . x+inv(x) = 0
remove inv
```

The semantics returns the following theory:

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
forall x,y,z:elem . x+0=x
                  . x+(y+z) = (x+y)+z
                  . x+inv(x) = 0
```

The module needs to be enlarged to the whole OMS.

Module Extraction applied to Groups (2)

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
forall x,y,z:elem . x+0=x
                  . x+(y+z) = (x+y)+z
                  . x+inv(x) = 0
                  . exists y:Elem . x+y=0
remove inv
```

The semantics returns the following theory:

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem
forall x,y,z:elem . x+0=x
                  . x+(y+z) = (x+y)+z
                  . exists y:Elem . x+y=0
```

Here, adding `inv` is conservative.

Approximation applied to Groups

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
forall x,y,z:elem . x+0=x
                  . x+(y+z) = (x+y)+z
                  . x+inv(x) = 0
forget inv
```

The semantics returns the following theory:

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem
forall x,y,z:elem . x+0=x
                  . x+(y+z) = (x+y)+z
                  . exists y:Elem . x+y=0
```

Computing finite interpolants can be hard, even undecidable.

Reduction applied to Groups

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
forall x,y,z:elem . x+0=x        . x+(y+z) = (x+y)+z
                  . x+inv(x)=0
hide inv
```

Semantics: class of all monoids that can be extended with an inverse, i.e. class of all groups. The effect is second-order quantification:

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem;
exists inv:Elem->Elem .
  forall x,y,z:elem . x+0=x
                      /\ x+(y+z) = (x+y)+z
                      /\ x+inv(x)=0
```

Filtering applied to Groups

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
forall x,y,z:elem . x+0=x
                  . x+(y+z) = (x+y)+z
                  . x+inv(x) = 0
reject inv
```

The semantics returns the following theory:

```
sort Elem
ops 0:Elem; __+__:Elem*Elem->Elem
forall x,y,z:elem . x+0=x
                  . x+(y+z) = (x+y)+z
```

Hide – Extract – Forget – Select

	hide/reveal	remove/extract	forget/keep	select/reject
semantic background	model reduct	conservative extension	uniform interpolation	theory filtering
relation to original	interpretable	subtheory	interpretable	subtheory
approach	model level	theory level	theory level	theory level
type of OMS	elusive	flattenable	flattenable	flattenable
signature of result	$= \Sigma$	$\geq \Sigma$	$= \Sigma$	$\geq \Sigma$
change of logic	possible	not possible	possible	not possible
application	specification	ontologies	ontologies	blending

Pros and Cons

	hide/reveal	remove/extract	forget/keep	select/reject
information loss	none	none	minimal	large
computability	depends	good/depends	depends	easy
signature of result	$= \Sigma$	$\geq \Sigma$	$= \Sigma$	$= \Sigma$
conceptual simplicity	simple (but unintuitive)	complex	farily simple	simple

Example for hiding: sorting

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
```

Correctness

Is insert sort correct w.r.t. the sorting specification?

```
interpretation correctness :
     { AbstractSort hide is_ordered, permutation }
  to { InsertSort hide insert }
end
```

Non-monotonicity

Non-monotonic Reasoning

Non-monotonic reasoning =
more premises may lead to fewer conclusions:
If b is a bird, it can fly.
But if b is a bird and a penguin, it cannot fly.

Non-monotonic reasoning is used in defeasible reasoning, default reasoning, abductive reasoning, belief revision, reasoning about subjective probabilities, ...

BUT: logical consequence $\Gamma \models_\Sigma \varphi$ is monotonic!

DOL's way of supporting non-monotonic reasoning:
closed-world assumptions

Closed-World Assumption

- Prop, FOL and OWL employ an open-world semantics
  1. predicates may hold for more individuals than specified in the theory
  2. a model may have more individuals than specified in the theory
  3. more equations than specified in the theory may hold between individuals
- sometimes, a closed-world semantics is useful
  1. predicates only hold for individuals if specified in the theory
  2. a model has only those individuals specified in the theory
  3. only equations specified in the theory hold between individuals
- Minimization (circumscription) addresses 1
- Freeness addresses 1-3
- Both are non-monotonic operations

Minimizations (circumscription)

- O_1 **then minimize {** O_2 **}**
- forces minimal interpretation of non-logical symbols in O_2

```
  Class: Block
  Individual: B1 Types: Block
  Individual: B2 Types: Block DifferentFrom: B1
then minimize {
        Class: Abnormal
        Individual: B1 Types: Abnormal }
then
  Class: Ontable
  Class: BlockNotAbnormal EquivalentTo:
    Block and not Abnormal SubClassOf: Ontable
then %implied
  Individual: B2 Types: Ontable
```

Minimizations

- O_1 **then minimize {** O_2 **}**
- forces minimal interpretation of non-logical symbols in O_2

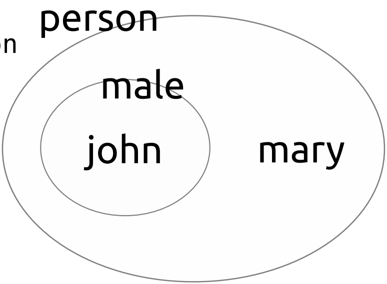
```
  Class: Block
  Individual: B1 Types: Block
  Individual: B2 Types: Block DifferentFrom: B1
then minimize {
        Class: Normal
        Individual: B2 Types: Normal }
then
  Class: Ontable SubClassOf: Block and Normal
then %implied
  Individual: B1 Types: not Ontable
```

Freeness

- free { O }
- O_1 then free { O }
- forces closed-world conditions 1-3

```
logic OWL
ontology Family_closed =
 free {
   Class: Person        Class: Male < Person
   Individual: john Types: Male
   Individual: mary Types: Person
  }
```

There is only one model
(up to isomorphism):



The Distributed Ontology, Model and Specification Language (DOL)
Day 4: Semantics of Structured OMS

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 3

On Day 3 we have looked at:

- Assembling OMS from pieces:
  Basic OMS, union, translation
- Making a large OMS smaller:
  module extraction, approximation, reduction, filtering
- Non-monotonic reasoning through employing
  a closed-world assumption:
  minimization, maximization, freeness, cofreeness

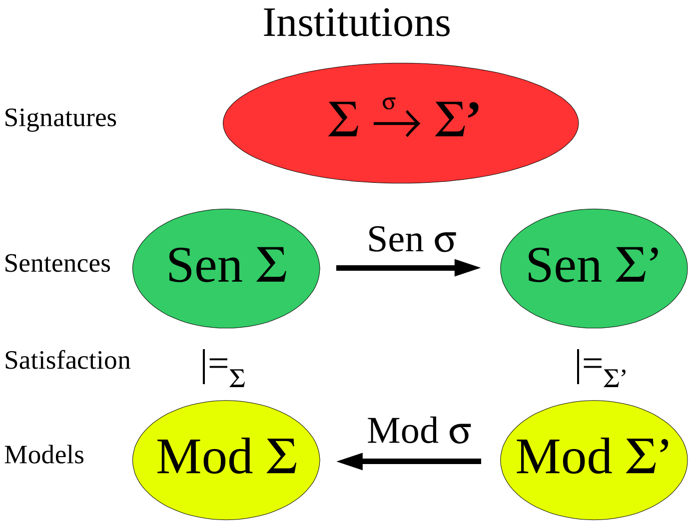
Today

We will focus today on:

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

Semantics of OMS

Institutions (intuition)



Some Basic Category Theory

Our use of category theory is modest, oriented towards providing easy proofs for very general results.

Definition (Category)

A category $\mathbf{C}$ is a graph together with a partial composition operation defined on edges that match:

if $f : A \to B$ and $g : B \to C$, then $f ; g : A \to C$.

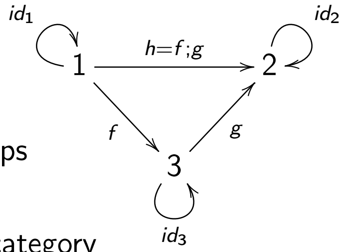
Graph nodes are called objects, graph edges are called morphisms.

Requirements on a category: morphisms behave monoid-like, that is,

- Composition has a neutral element $id_A : A \to A$ (for each object $A \in |\mathbf{C}|$):
  for $f : A \to B$, $id_A ; f = f$ and $f ; id_B = f$
- Composition is associative:
  $(f ; g) ; h = f ; (g ; h)$ if both sides are defined

Categories: Examples

- sets and functions
- FOL signatures and signature morphisms
- OWL signatures and signature morphisms
- logical theories and theory morphisms
- groups and group homomorphisms
- general algebras and homomorphisms
- metric spaces and contractions
- topological spaces and continuous maps
- automata and simulations
- each pre-order, seen as a graph, is a category
- each monoid is a category with one object



Opposite Categories

Definition (Opposite category)

Given a category $\mathbf{C}$, its opposite category $\mathbf{C}^{op}$ has the same objects and morphism as $\mathbf{C}$, but with all morphisms reversed. That is,

$$\text{if } f\colon A \to B \in \mathbf{C}, \text{ then } f\colon B \to A \in \mathbf{C}^{op}.$$

$$\text{if } f;g = h \text{ in } \mathbf{C}, \text{ then } g;f = h \text{ in } \mathbf{C}^{op}.$$

Functors

Definition (Functor)

Given categories $\mathbf{C}_1$ and $\mathbf{C}_2$, a functor $F\colon \mathbf{C}_1 \to \mathbf{C}_2$ is a graph homomorphism $F\colon \mathbf{C}_1 \to \mathbf{C}_2$ preserving the monoid structure, that is

- Neutral elements are preserved:
  $$F(id_A) = id_{F(A)}$$
  for each object $A \in |\mathbf{C}|$
- Composition is preserved:
  $$F(f;g) = F(f); F(g)$$
  for each $f\colon A \to B$, $g\colon B \to C \in \mathbf{C}$.

Institutions (formal definition)

An institution $\mathcal{I} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ consists of:

- a category **Sign** of signatures;
- a functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$, giving a set $\mathbf{Sen}(\Sigma)$ of Σ-sentences for each signature $\Sigma \in |\mathbf{Sign}|$, and a function $\mathbf{Sen}(\sigma)\colon \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$ that yields σ-translation of Σ-sentences to Σ'-sentences for each $\sigma\colon \Sigma \to \Sigma'$;
- a functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$, giving a category $\mathbf{Mod}(\Sigma)$ of Σ-models for each signature $\Sigma \in |\mathbf{Sign}|$, and a functor $\_|_\sigma = \mathbf{Mod}(\sigma)\colon \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$; for each $\sigma\colon \Sigma \to \Sigma'$;
- for each $\Sigma \in |\mathbf{Sign}|$, a satisfaction relation $\models_{\mathcal{I},\Sigma} \subseteq \mathbf{Mod}(\Sigma) \times \mathbf{Sen}(\Sigma)$

such that for any signature morphism $\sigma\colon \Sigma \to \Sigma'$, Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$ and Σ'-model $M' \in \mathbf{Mod}(\Sigma')$:

$$M' \models_{\mathcal{I},\Sigma'} \sigma(\varphi) \text{ iff } M'|_\sigma \models_{\mathcal{I},\Sigma} \varphi \qquad \text{[Satisfaction condition]}$$

Sample Institutions

- Prop, FOL and OWL are institutions
  we have proven the satisfaction conditions in lecture 2

Plenty of Institutions

- Lary Moss' logics from his ESSLLI evening talk on Tuesday
- first-order, higher-order logic, polymorphic logics
- logics of partial functions
- modal logic (epistemic logic, deontic logic, description logics, logics of knowledge and belief, agent logics)
- μ-calculus, dynamic logic
- spatial logics, temporal logics, process logics, object logics
- intuitionistic logic
- linear logic, non-monotonic logics, fuzzy logics
- paraconsistent logic, database query languages

Working in an Arbitrary Logical System

Many notions and results generalise to an arbitrary institution:

- logical consequence
- logical theory
- satisfiability
- conservative extension
- theory morphism
- many more . . .

In the sequel, fix an arbitrary instution I.

Weakly inclusive institutions

Definition (adopted from Goguen, Roșu)

A weakly inclusive category is a category having a singled out class of morphisms (called inclusions) which is closed under identities and composition. Inclusions hence form a partial order.
An weakly inclusive institution is one with an inclusive signature category such that

- the sentence functor preserves inclusions,
- the inclusion order has a least element (denote $\emptyset$), suprema (denoted $\cup$), infima (denoted $\cap$), and differences (denoted $\setminus$),
- model categories are weakly inclusive.

$M|_\Sigma$ means $M|_\iota$ where $\iota : \Sigma \to Sig(M)$ is the inclusion.
In the sequel, fix an arbitrary weakly inclusive instution I.

Semantic domains for OMS in DOL

Flattenable OMS (can be flattened to a basic OMS)

- basic OMS
- extensions, unions, translations
- approximations, module extractions, filterings
- semantics: (Σ, Ψ) (theory-level)
  - Σ: a signature in I, also written $Sig(O)$
  - Ψ: a set of Σ-sentences, also written $Th(O)$

Elusive OMS (= non-flattenable OMS)

- reductions, minimization, maximization, (co)freeness (elusive)
- semantics: $(\Sigma, \mathcal{M})$ (model-level)
  - Σ: a signature in I, also written $Sig(O)$
  - $\mathcal{M}$: a class of Σ-models, also written $Mod(O)$

We can obtain the model-level semantics from the theory-level semantics by taking $\mathcal{M} = \{M \in \mathbf{Mod}(\Sigma) \mid M \models \Psi\}$.

Semantics of basic OMS

We assume that $[\![O]\!]_{basic} = (\Sigma, \Psi)$ for some OMS language based on I. The semantics consists of

- a signature Σ in I
- a set Ψ of Σ-sentences

This direct leads to a theory-level semantics for OMSx:

$$[\![O]\!]^T_\Gamma = [\![O]\!]_{basic}$$

Generally, if a theory-level semantics is given: $[\![O]\!]^T_\Gamma = (\Sigma, \Psi)$, this leads to a model-level semantics as well:

$$[\![O]\!]^M_\Gamma = (\Sigma, \{M \in Mod(\Sigma) \mid M \models \Psi\})$$

Semantics of extensions

O_1 flattenable $[\![O_1 \textbf{ then } O_2]\!]^T_\Gamma = (\Sigma_1 \cup \Sigma_2, \Psi_1 \cup \Psi_2)$

where

- $[\![O_1]\!]^T_\Gamma = (\Sigma_1, \Psi_1)$
- $[\![O_2]\!]_{basic} = (\Sigma_2, \Psi_2)$

O_1 elusive $[\![O_1 \textbf{ then } O_2]\!]^M_\Gamma = (\Sigma_1 \cup \Sigma_2, \mathcal{M}')$

where

- $[\![O_1]\!]^M_\Gamma = (\Sigma_1, \mathcal{M}_1)$
- $[\![O_2]\!]_{basic} = (\Sigma_2, \Psi_2)$
- $\mathcal{M}' = \{M \in \textbf{Mod}(\Sigma_1 \cup \Sigma_2) \mid M \models \Psi_2, M|_{\Sigma_1} \in \mathcal{M}_1\}$

Semantics of extensions (cont'd)

%mcons (%def, %mono) leads to the additional requirement that

each model in $\mathcal{M}_1$ has a (unique, unique up to isomorphism) $\Sigma_1 \cup \Sigma_2$-expansion to a model in $\mathcal{M}'$.

%implies leads to the additional requirements that

$\Sigma_2 \subseteq \Sigma_1$ *and* $\mathcal{M}' = \mathcal{M}_1$.

%ccons leads to the additional requirement that

$\mathcal{M}' \models \varphi$ *implies* $\mathcal{M}_1 \models \varphi$ *for any* Σ_1*-sentence* φ.

Theorem

%mcons implies %ccons, but not vice versa.

References to Named OMS

- Reference to an OMS existing on the Web
- written directly as a URL (or IRI)
- Prefixing may be used for abbreviation

`http://owl.cs.manchester.ac.uk/co-ode-files/ontologies/pizza.owl`

`co-ode:pizza.owl`

Semantics Reference to Named OMS: $[\![iri]\!]_\Gamma = \Gamma(iri)$
where Γ is a global map of IRIs to OMS denotations

Semantics of unions

O_1, O_2 flattenable $[\![O_1 \textbf{ and } O_2]\!]^T_\Gamma = (\Sigma_1 \cup \Sigma_2, \Psi_1 \cup \Psi_2)$, where

- $[\![O_i]\!]^T_\Gamma = (\Sigma_i, \Psi_i)$ $(i = 1, 2)$

one of O_1, O_2 elusive $[\![O_1 \textbf{ and } O_2]\!]^M_\Gamma = (\Sigma_1 \cup \Sigma_2, \mathcal{M})$, where

- $[\![O_i]\!]^M_\Gamma = (\Sigma_i, \mathcal{M}_i)$ $(i = 1, 2)$
- $\mathcal{M} = \{M \in \textbf{Mod}(\Sigma_1 \cup \Sigma_2) \mid M|_{\Sigma_i} \in \mathcal{M}_i, i = 1, 2\}$

Semantics of translations

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

$$[\![O \textbf{ with } \sigma : \Sigma \to \Sigma']\!]^T_\Gamma = (\Sigma', \sigma(\Psi))$$

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

$$[\![O \textbf{ with } \sigma : \Sigma \to \Sigma']\!]^M_\Gamma = (\Sigma', \mathcal{M}')$$

where $\mathcal{M}' = \{M \in \mathbf{Mod}(\Sigma') \mid M|_\sigma \in \mathcal{M}\}$

Hide – Extract – Forget – Select

	hide/reveal	remove/extract	forget/keep	select/reject
semantic background	model reduct	conservative extension	uniform interpolation	theory filtering
relation to original	interpretable	subtheory	interpretable	subtheory
approach	model level	theory level	theory level	theory level
type of OMS	elusive	flattenable	flattenable	flattenable
signature of result	$= \Sigma$	$\geq \Sigma$	$= \Sigma$	$\geq \Sigma$
change of logic	possible	not possible	possible	not possible
application	specification	ontologies	ontologies	blending

Semantics of reductions

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

- $[\![O \textbf{ reveal } \Sigma']\!]_\Gamma^M = (\Sigma', \mathcal{M}|_{\Sigma'})$, where $\mathcal{M}|_{\Sigma'} = \{M|_{\Sigma'} \mid M \in \mathcal{M}\})$
- $[\![O \textbf{ hide } \Sigma']\!]_\Gamma^M = [\![O \textbf{ reveal } \Sigma \setminus \Sigma']\!]_\Gamma^M$

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

Modules

Definition

$O' \subseteq O$ is a Σ-module of (flat) O iff O is a model-theoretic Σ-conservative extension of O', i.e. for every model M of O', $M|_\Sigma$ can be expanded to an O-model.

Depleting modules

Definition

Let O_1 and O_2 be two OMS and $\Sigma \subseteq Sig(O_i)$.
Then O_1 and O_2 are Σ-inseparable ($O_1 \equiv_\Sigma O_2$) iff

$$Mod(O_1)|_\Sigma = Mod(O_2)|_\Sigma$$

Definition

$O' \subseteq O$ is a depleting Σ-module of (flat) O iff $O \setminus O' \equiv_{\Sigma \cup Sig(O')} \emptyset$.

Theorem

1. *Depleting Σ-modules are Σ-conservative.*
2. *The minimum depleting Σ-module always exists.*

Semantics of module extraction (remove/extract)

Note: O must be flattenable!

Let $[\![O]\!]_\Gamma^T = (\Sigma, \Psi)$.
$[\![O\ \textbf{extract}\ \Sigma_1]\!]_\Gamma^T = (\Sigma_2, \Psi_2)$
where $(\Sigma_2, \Psi_2) \subseteq (\Sigma, \Psi)$ is the minimum depleting Σ_1-module of (Σ, Ψ)

$[\![O\ \textbf{remove}\ \Sigma_1]\!]_\Gamma^T = [\![O\ \textbf{extract}\ \Sigma \setminus \Sigma_1]\!]_\Gamma^T$

Tools can extract other types of module though (i.e. using locality). However, any two modules will have the same Σ-consequences.

Semantics of interpolation (forget/keep)

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

$[\![O \text{ **keep in** } \Sigma']\!]_{\Gamma}^{T} = (\Sigma', \{\varphi \in \mathbf{Sen}(\Sigma') \mid \Psi \models \varphi\})$
Note: any logically equivalent theory will also do).
Challenge: find a finite theory (= uniform interpolant). This is not always possible, and sometimes theoretically possible but not computable.

$[\![O \text{ **forget** } \Sigma']\!]_{\Gamma}^{T} = [\![O \text{ **keep in** } \Sigma \setminus \Sigma']\!]_{\Gamma}^{T}$

Semantics of select/reject

Note: O must be flattenable!

Let $[\![O]\!]_\Gamma^T = (\Sigma, \Psi)$.

$[\![O \textbf{ select } (\Sigma', \Phi)]\!]_\Gamma^T = (\Sigma, Sen(\iota)^{-1}(\Psi) \cup \Phi)$
where $\iota : \Sigma' \to \Sigma$ is the inclusion

$[\![O \textbf{ reject } (\Sigma', \Phi)]\!]_\Gamma^T = (\Sigma \setminus \Sigma', Sen(\iota)^{-1}(\Psi) \setminus \Phi)$
where $\iota : \Sigma \setminus \Sigma' \to \Sigma$ is the inclusion

Relations among the different notions

$$
\begin{array}{ll}
 & Mod(O\ \textbf{reveal}\ \Sigma) \\
= & Mod(O\ \textbf{extract}\ \Sigma)|_{Sig(O)\setminus\Sigma} \\
\subseteq & Mod(O\ \textbf{keep}\ \Sigma) \\
\subseteq & Mod(O\ \textbf{select}\ \Sigma)
\end{array}
$$

Semantics of minimizations

Let $[\![O_1]\!]_\Gamma^M = (\Sigma_1, \mathcal{M}_1)$
Let $[\![O_1 \text{ **then** } O_2]\!]_\Gamma^M = (\Sigma_2, \mathcal{M}_2)$
Then

$$[\![O_1 \textbf{ then minimize } O_2]\!]_\Gamma^M = (\Sigma_2, \mathcal{M})$$

where
$\mathcal{M} = \{M \in \mathcal{M}_2 \mid M \text{ is minimal in } \{M' \in \mathcal{M}_2 \mid M'|_{\Sigma_1} = M|_{\Sigma_1}\}\}$

Note that in a weakly inclusive institution, inclusion model morphisms provide a partial order on models.

Dually: maximization.

Initial Objects

Definition

An object I in a category $\mathbf{C}$ is called an initial object, if for each object $A \in |\mathbf{C}|$, there is a unique morphism $I \rightarrow A$.

Example

Initital objects in different categories:

- sets and functions: the empty set
- FOL signatures: the empty signature
- algebras and homomorphisms: the term algebra
- models of Horn clauses: the Herbrand model

Theorem

Initial objects are unique up to isomorphism.

Semantics of freeness

We only treat the special case of **free** $\{O\}$.

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

$$[\![\textbf{free } O]\!]_\Gamma^M = (\Sigma, \{M \in \mathcal{M} \mid M \text{ is initial in } \mathcal{M}\})$$

Semantics of interpretations

Let $[\![O_i]\!]_\Gamma^M = (\Sigma_i, \mathcal{M}_i)$ $(i = 1, 2)$

$$[\![\textbf{interpretation } IRI : O_1 \textbf{ to } O_2 = \sigma]\!]_\Gamma^M$$

is defined iff

$$Mod(\sigma)(\mathcal{M}_2) \subseteq \mathcal{M}_1$$

Note that this is the same condition as for theory morphisms.

Proof calculus

Logical Consequences and Refinement of OMS

Definition (Logical Consequences of an OMS)

$$O \models_\Sigma \varphi \qquad \text{iff} \qquad \Sigma = Sig(O),\ M \models_\Sigma \varphi \text{ for all } M \in Mod(O)$$

Definition (Refinement between two OMS)

$$O \rightsquigarrow O' \qquad \text{iff} \qquad Mod(O') \subseteq Mod(O)$$

Entailment systems

Definition

Given an institution $\mathcal{I} = (\mathbf{Sign}, \mathbf{Sen}, \mathit{Mod}, \models)$, an entailment system $\vdash$ for $\mathcal{I}$ consists of relations $\vdash_\Sigma \subseteq \mathcal{P}(\mathbf{Sen}(\Sigma)) \times \mathbf{Sen}(\Sigma)$ such that

1. reflexivity: for any $\varphi \in \mathbf{Sen}(\Sigma)$, $\{\varphi\} \vdash_\Sigma \varphi$,
2. monotonicity: if $\Gamma \vdash_\Sigma \varphi$ and $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash_\Sigma \varphi$,
3. transitivity: if $\Gamma \vdash_\Sigma \varphi_i$ for $i \in I$ and $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash_\Sigma \psi$, then $\Gamma \vdash_\Sigma \psi$,
4. $\vdash$-translation: if $\Gamma \vdash_\Sigma \varphi$, then for any $\sigma : \Sigma \longrightarrow \Sigma'$ in $\mathbf{Sign}$, $\sigma(\Gamma) \vdash_{\Sigma'} \sigma(\varphi)$,
5. soundness: if $\Gamma \vdash_\Sigma \varphi$ then $\Gamma \models_\Sigma \varphi$.

The entailment system is complete if, in addition, $\Gamma \models_\Sigma \varphi$ implies $\Gamma \vdash_\Sigma \varphi$.

Proof calculus for entailment (Borzyszkowski) covering some part of DOL

$$(CR)\ \frac{\{O \vdash \varphi_i\}_{i \in I} \quad \{\varphi_i\}_{i \in I} \vdash \varphi}{O \vdash \varphi} \qquad (basic)\ \frac{\varphi \in \Gamma}{\langle \Sigma, \Gamma \rangle \vdash \varphi}$$

$$(sum1)\ \frac{O_1 \vdash \varphi}{O_1 \ \textbf{and}\ O_2 \vdash \varphi} \qquad (sum2)\ \frac{O_2 \vdash \varphi}{O_2 \ \textbf{and}\ O_2 \vdash \varphi}$$

$$(trans)\ \frac{O \vdash \varphi}{O \ \textbf{with}\ \sigma \vdash \sigma(\varphi)} \qquad (derive)\ \frac{O \vdash \sigma(\varphi)}{O \ \textbf{hide}\ \sigma \vdash \varphi}$$

Soundness means: $O \vdash \varphi$ implies $O \models \varphi$
Completeness means: $O \models \varphi$ implies $O \vdash \varphi$

Proof calculus for refinement (Borzyszkowski)

$$(Basic)\ \frac{O \vdash \Gamma}{\langle \Sigma, \Gamma \rangle \rightsquigarrow O} \qquad (Sum)\ \frac{O_1 \rightsquigarrow O \quad O_2 \rightsquigarrow O}{O_1 \textbf{ and } O_2 \rightsquigarrow O}$$

$$(Trans)\ \frac{O \rightsquigarrow O' \textbf{ hide } \sigma}{O \textbf{ with } \sigma \rightsquigarrow O'}$$

$$(Derive)\ \frac{O \rightsquigarrow O''}{O \textbf{ hide } \sigma \rightsquigarrow O'}$$

if $\sigma : O' \longrightarrow O''$
is a conservative extension

Soundness means: $O_1 \rightsquigarrow O_2$ implies $O_1 \rightsquigarrow O_2$

Completeness means: $O_1 \rightsquigarrow O_2$ implies $O_1 \rightsquigarrow O_2$

Soundness and Completeness

Theorem (Borzyszkowski, Tarlecki, Diaconescu)

The calculi for structured entailment and refinement are sound.
Under the assumptions that

- *the institution admits Craig-Robinson interpolation,*
- *the institution has weak model amalgamation, and*
- *the entailment system is complete,*

the calculi are also complete.

For refinement, we need an oracle for conservative extensions.
Craig-Robinson interpolation, weak model amalgamation:
technical model-theoretic conditions

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:

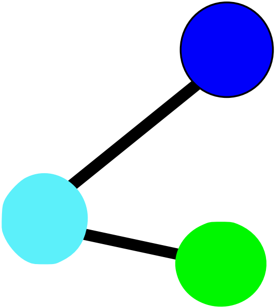
- Semantics of structured OMS
  - based on institutions
- Proofs in OMS
  - based on entailment systems

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.

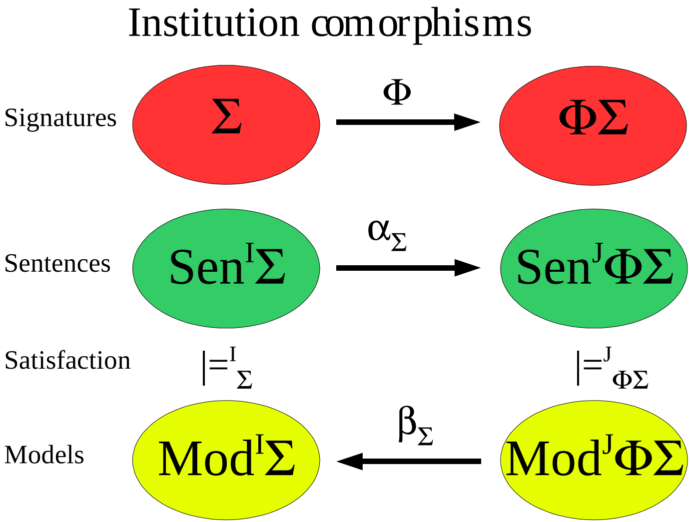
OWL gives better tool support, FOL greater expressiveness.

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 $M' \in \mathbf{Mod}'(\Phi(\Sigma))$:

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

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

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):

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

Translation of sentences

- $\alpha_\Sigma(C \sqsubseteq D) = \forall x\colon \mathit{Thing}.\,(\alpha_x(C) \rightarrow \alpha_x(D))$
- $\alpha_\Sigma(a : C) = \alpha_x(C)[x \mapsto a]$[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 $M' \in \text{Mod}^{FOL}(\Phi(\Sigma))$ 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

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

Proof.

By induction over the structure of C.

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

The satisfaction condition now follows easily.

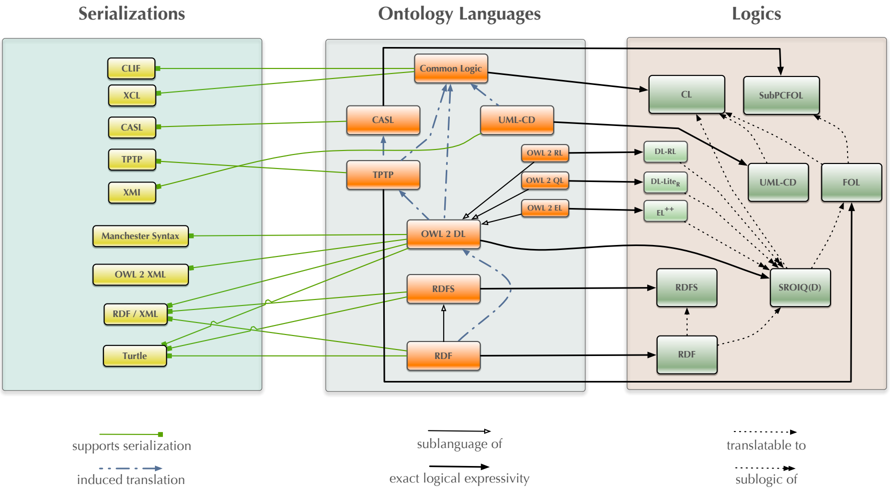
Heterogeneous logical environments

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

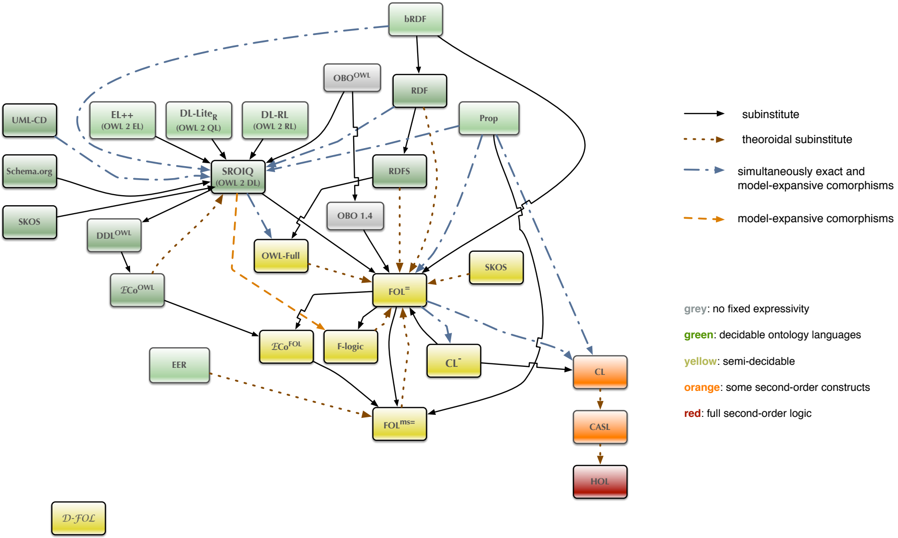
- a logic graph, consisting of institutions, institution comorphisms (translations) and institution morphisms (projections, see below),
- an OMS language graph, and
- support relations.

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

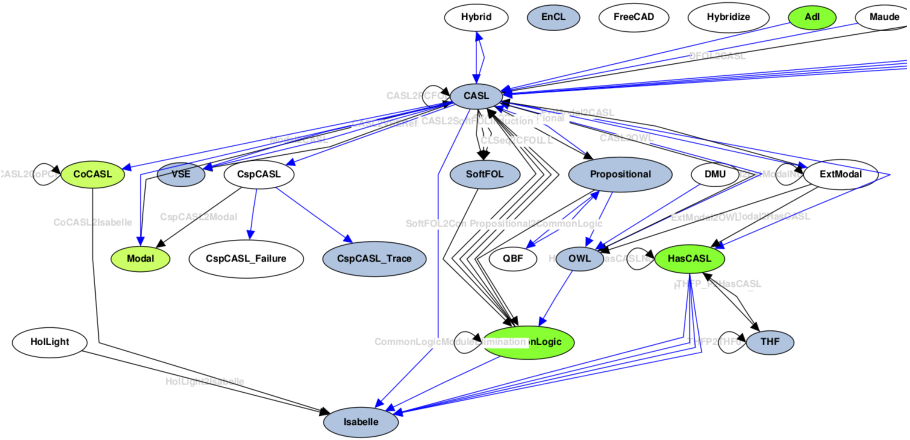
Moreover, for each language we have a default selection of a logic and a serialization. There are also default translations.


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

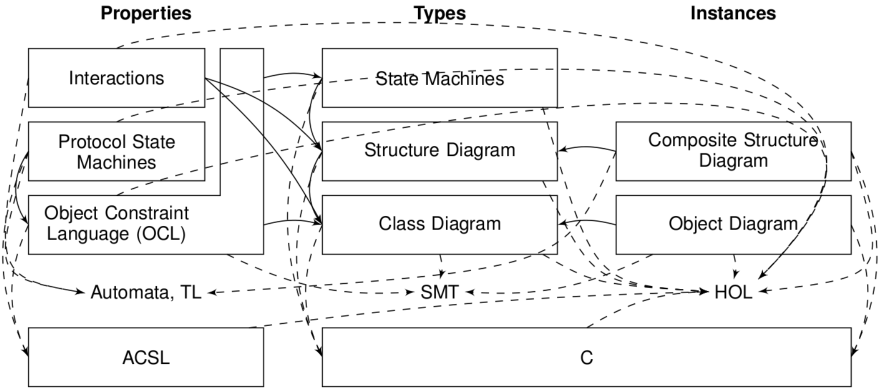
Ontologies: An Initial Logic Graph



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

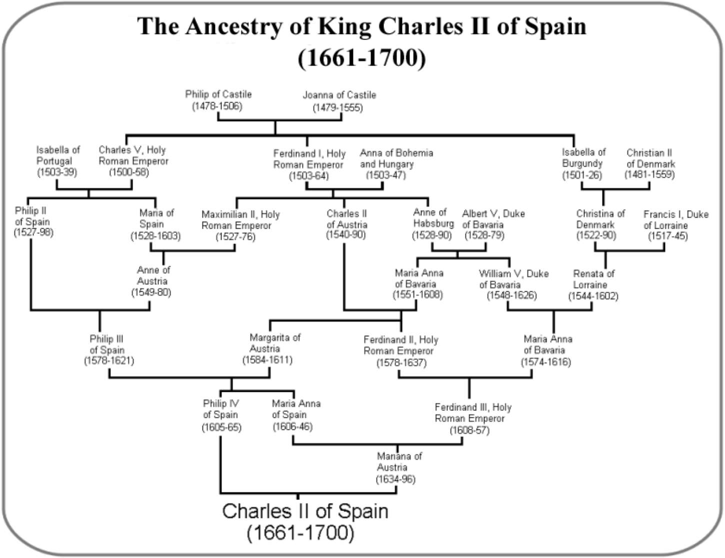
Extended task

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 → 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

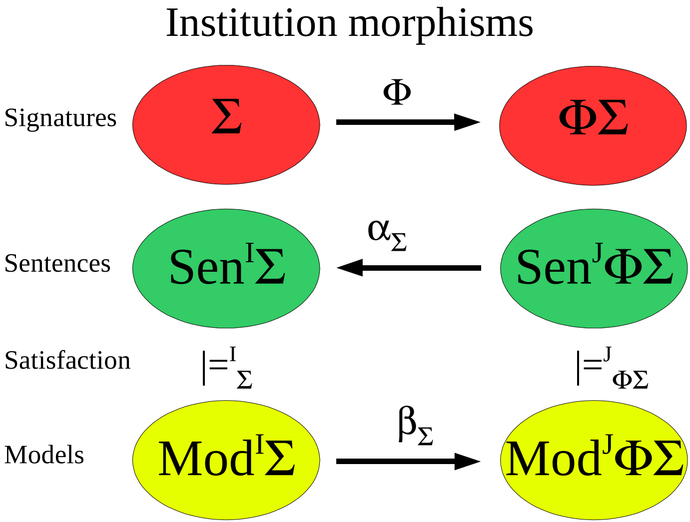
```
ontology CharlesII_ABox =
  Individual: CharlesII ... %% Charles II ABox
end

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

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

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

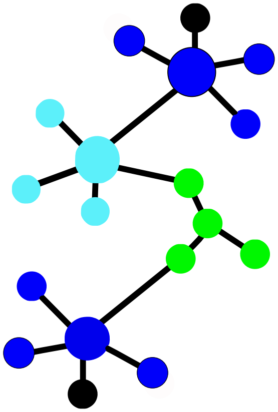
$\mathcal{M}|_{\Sigma'}$ may be impossible to capture by a theory (even if $\mathcal{M}$ is).

Semantics of heterogeneous approximation

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

- homogeneous approximation
  $[\![O \textbf{ keep in } \Sigma']\!]^T_\Gamma = (I, \Sigma', \{\varphi \in \textbf{Sen}(\Sigma') \mid \Psi \models \varphi\})$
  (note: any logically equivalent theory will also do)
  $[\![O \textbf{ forget } \Sigma']\!]^T_\Gamma = [\![O \textbf{ keep in } \Sigma \setminus \Sigma']\!]^T_\Gamma$
- heterogeneous approximation
  $[\![O \textbf{ keep in } \Sigma' \textbf{ with } I']\!]^T_\Gamma =$
  $(I', \Sigma', \{\varphi \in \textbf{Sen}^{I'}(\Sigma') \mid \Psi \models \rho^{Sen}(\varphi)\})$
  where $\rho : I' \to I$ is the inclusion
  and Σ' is such that $\rho^{Sig}(\Sigma') \subseteq \Sigma$
  $[\![O \textbf{ forget } \Sigma' \textbf{ with } I']\!]^T_\Gamma = [\![O \textbf{ keep in } \Sigma \setminus \Sigma' \textbf{ with } I']\!]^T_\Gamma$

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

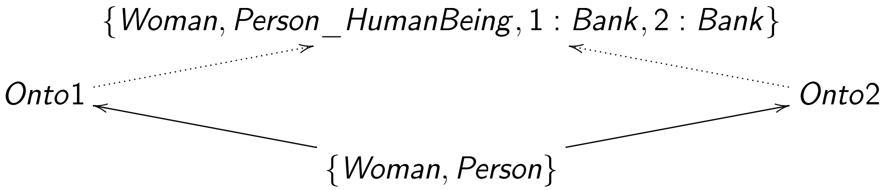
- **combine** N
- N is a network
- semantics is the (a) colimit of the diagram N

```
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



Alignments

- **alignment** *Id* $card_1$ $card_2$: O_1 **to** O_2 $= c_1, \dots c_n$
  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

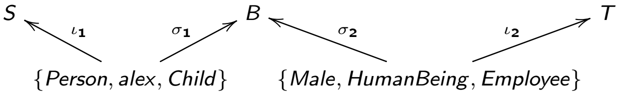
The resulting diagram N includes (institution-specific) W-alignment diagrams for each alignment A_i. Using **assuming**, assumptions about the domains of all OMS can be specified:

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



where

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

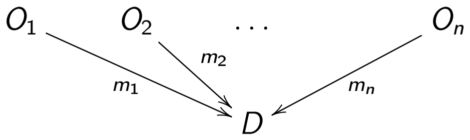
Resulting colimit

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



- 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

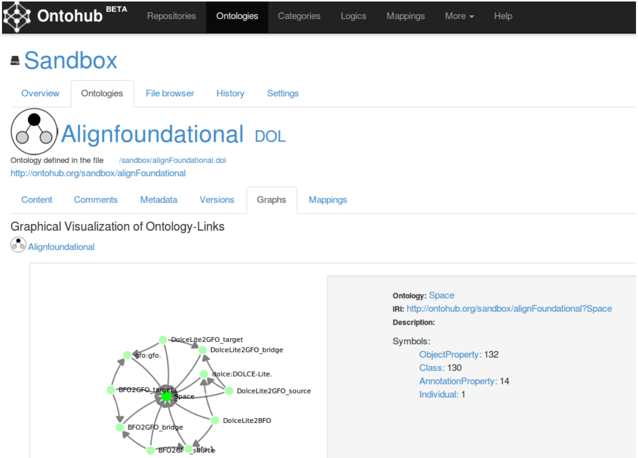
```
%prefix(   gfo: <http://www.onto-med.de/ontologies/>
           dolce: <http://www.loa-cnr.it/ontologies/>
           bfo: <http://www.ifomis.org/bfo/>            )%

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
```

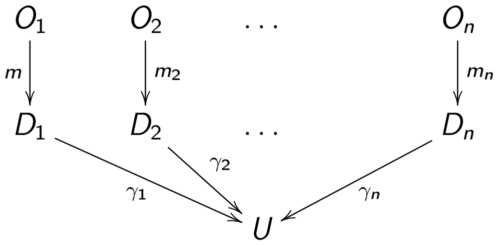
Alignment of Upper Ontologies — Combination

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



Integrated semantics of diagrams

Framework: different domains reconciled in a global domain



- 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}

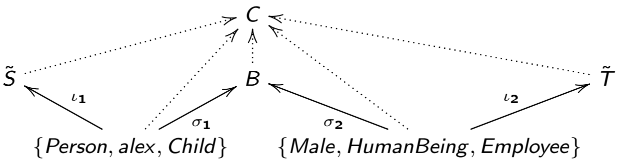
Relativization of an OWL ontology

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



where

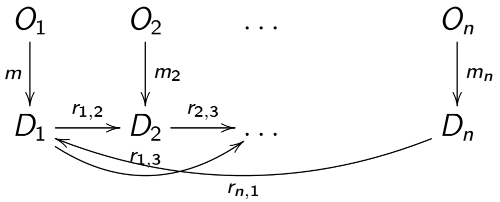
ontology B =
Class: *Thing*$_S$ **Class:** *Thing*$_T$
Class: *Person_HumanBeing* **SubClassOf:** *Thing*$_S$, *Thing*$_T$
Class: *Male* **Class:** *Employee*
Class: *Child* **SubClassOf:** *Thing*$_T$ **and** $\neg$ *Employee*
Individual: *alex* **Types:** *Male*

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

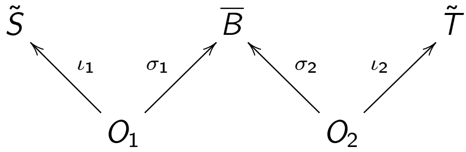
Framework: different domains related by coherent relations



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



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

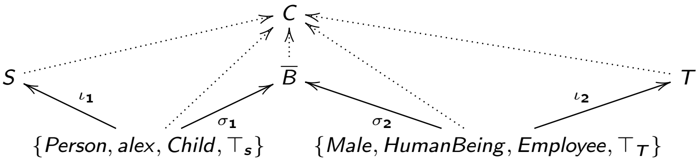
- r_{ij} are added to $\overline{B}$ as roles with domain $\top_S$ and range $\top_T$
- the correspondences are translated to axioms involving these roles:
  - $s_i = t_j$ becomes $s_i \ r_{ij} \ t_j$
  - $a_i \in c_j$ becomes $a_i \in \exists r_{ij}.c_j$
  - ...
- the properties of the roles are added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



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

$$O_1 \rightsquigarrow O_2$$

Refinements

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)

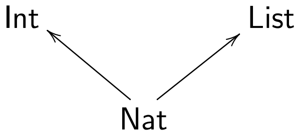
```
spec NatBin =
generated type Bin ::= 0 | 1 | __0(Bin) | __1(Bin)
ops __+__ , __++__ : Bin * Bin -> Bin
forall x, y : Bin
 .  0 0 = 0  .  0 1 = 1
 .  not  (0 = 1)  .  x 0 = y 0 => x = y .  not  (x 0 = y 1)  .
x 1 = y 1 => x = y
 .  0 + 0 = 0  .  0 ++ 0 = 1
 .  x 0 + y 0 = (x + y) 0  .  x 0 ++ y 0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

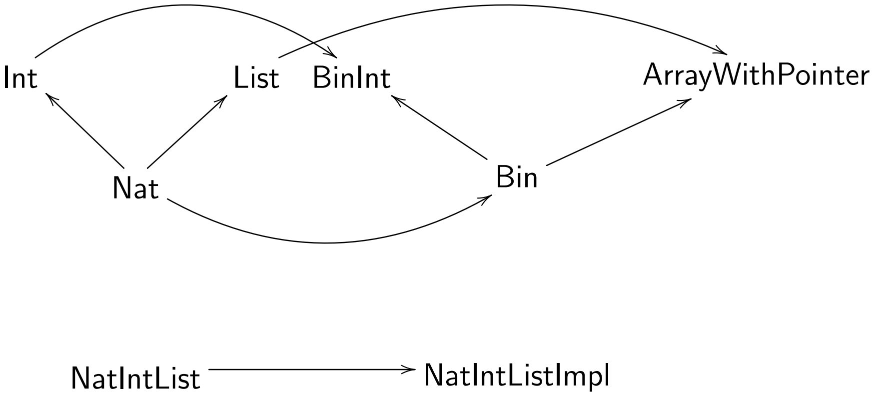
```
spec Nat = ...
end
spec Int = Nat then ...
end
spec List = Nat then ...
end
network NatIntList = Nat, Int, List
end
```



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`

Feedback?