Generalized Sketches and Indexed vs. Fibred Semantics

Uwe Wolter

University of Bergen, Norway

IMAR, November 18th 2009, Bucharest, Romania

Outline

Model Driven Engineering (MDE)

State of the Art

- Diagrammatic, i.e., graph-based, modeling techniques like UML-diagrams, ER-diagrams, ORM-diagrams, database schemata, ontologies, XML,. . . , are widely used in software engineering (SE).
- Until recently diagrammatic models were mainly used as a communication medium between business experts, software designers and programmers, and their precision and semantics was (although very desirable) not a must.

Model Driven Engineering (MDE)

Dramatical changes

Few years ago started a rapid invasion of the model-centric trends in software industry (MDE, MDD, MDA$^{\mathrm{TM}}$). These activities aim at making models rather than code the primary artifacts of software development with code to be generated directly from models.

New needs

For MDE and friends, having a precise formal semantics for the diagrammatic notations they employ is an undisputable must. An overwhelming amount of formal semantics for diagrammatic languages in use has been proposed. Most of them employ the familiar first-order or similar logical systems, and fail to do the job because of unfortunate mismatch between the logical machineries they use for formalization (string-based) and the internal logics of the domains they intend to formalize (graph-based).

Graph-based specifications?

Question

Does there exist something in mathematics that could be called a "graph-based specification formalism"?

Answer

Yes, the sketch formalism developed within Category Theory (CT).

- Introduced 1968 by Ehresmann.
- Re-formulated and re-launched in the 80s by Barr and Wells for application in computer science (CS).
- Extremely expressive.

Sketches

Definition of sketch

A **sketch** $\mathcal{S} = (G^{\mathcal{S}}, \Delta^{\mathcal{S}})$ consists of a graph $G^{\mathcal{S}}$ and a set $\Delta^{\mathcal{S}}$ of **diagrams** (d, I) in $G^{\mathcal{S}}$, i.e., of graph homomorphisms $d : I \longrightarrow G^{\mathcal{S}}$. Moreover, each diagram in $\Delta^{\mathcal{S}}$ is labeled either as a commutative diagram or a limit cone or a colimit cocone.

Remark

Sketches can be classified according to the structure of the **shapes** I and the labels used. Examples are finite product sketches and finite limit sketches (left exact sketches).

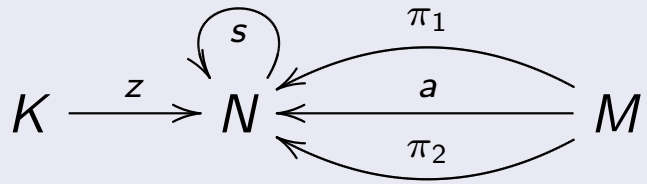
Sketches - Example "Algebraic Signatures"

Algebraic signature for natural numbers

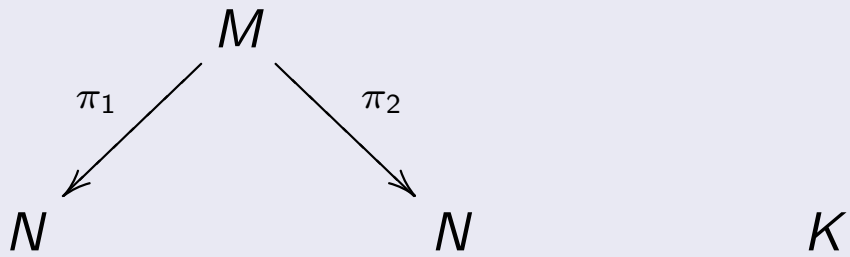
We declare one sort N and three operations $z : \longrightarrow N$, $s : N \longrightarrow N$, $a : N\,N \longrightarrow N$.

The corresponding finite product sketch

Underlying graph:

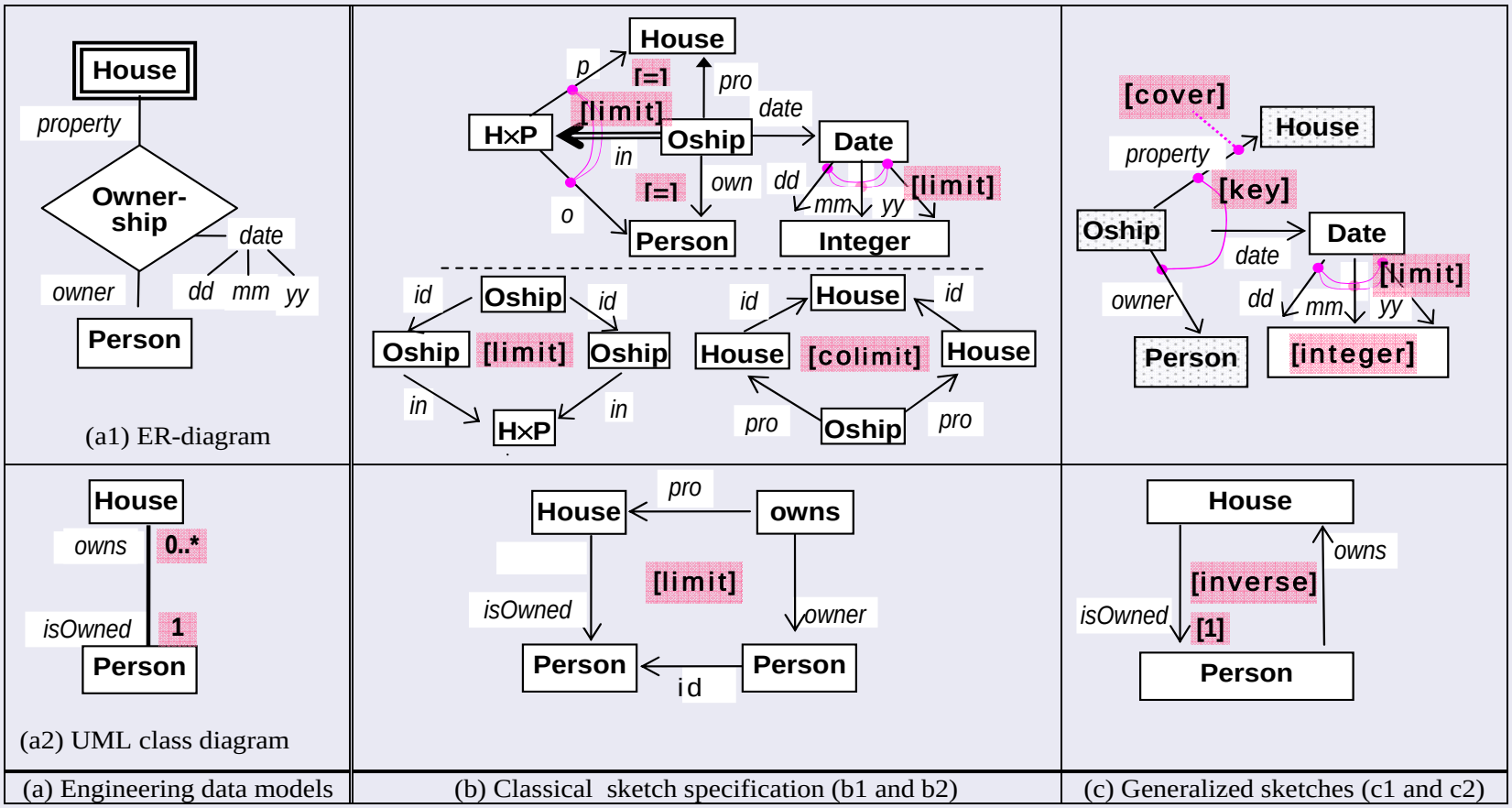


Product cones:



Sketches - Example Diagrammatic Modeling

A sample of sketching a diagrammatic notation



Why Generalized Sketches?

Inherent drawbacks of sketches in practical applications

- Auxiliary elements have to be introduced.
- Some original elements can be only derived.
- Simple and important properties like mono, epi have to be coded.

$\Rightarrow$ For software engineers this looks artificial, unnecessary, and misleading.

- There are practical relevant predicates that can not be expressed by limits and colimits, at all. Bear in mind that the categories **Rel** and **Mult**, respectively, have no limits.

Why Generalized Sketches?

Some History

Generalized sketches have been developed in the 90's in parallel by Makkai and a group in Latvia (Diskin, Cadish, et.al.)

- Makkai developed an abstract categorical approach to completeness theorems in logic.
- The Latvian group developed generalized sketches in the context of data modeling and applied them in a few industrial projects.

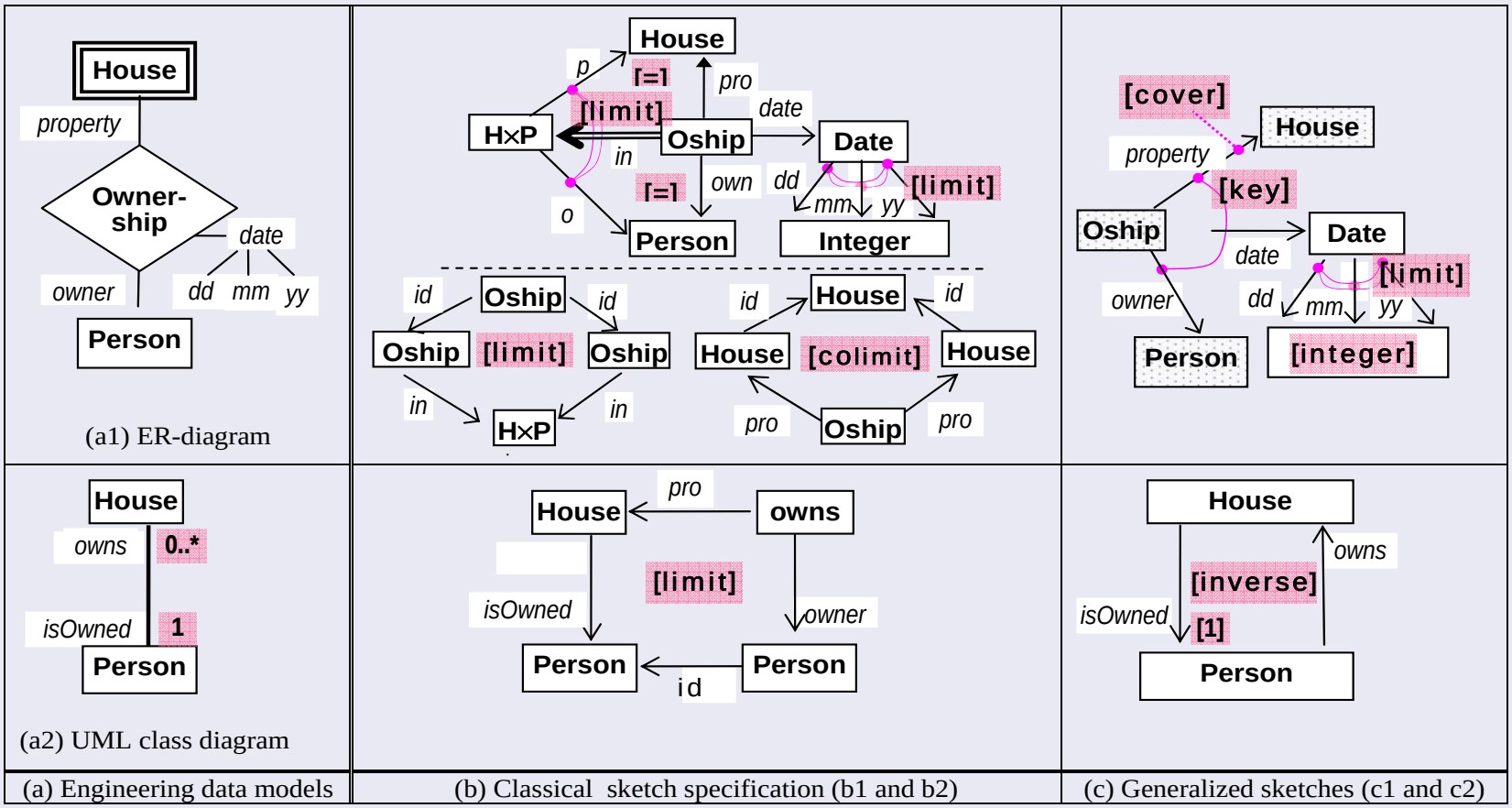
General in two aspects

1. Arbitrary "base category" **Base** instead of category **Graph**.
2. Arbitrary "user defined" predicates instead of "commutative", "limit", "colimit" only.

Attention: The semantics of the predicates "commutative", "limit", "colimit" is pre-defined in any category. In contrast, the semantics of a "user defined" predicate has to be defined explicitly by its inventor.

Generalized Sketches - Example Diagrammatic Modeling

A sample of sketching a diagrammatic notation



Generalized Sketches

Definition of Notation

A **(diagram predicate) notation** $\mathcal{N} = (\Pi, \alpha)$ over a category **Base** is given by a set Π of predicate symbols and a map $\alpha : \Pi \to \mathbf{Base}_0$. For a predicate symbol $P \in \Pi$, the **Base**-object $\alpha(P)$ is called the **arity** of P.

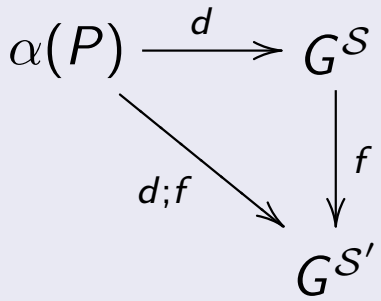
Definition of Specification

An $\mathcal{N}$-**specification** ($\mathcal{N}$-**sketch**) $\mathcal{S} = (G^{\mathcal{S}}, \Delta^{\mathcal{S}})$ consists of a **Base**-object $G^{\mathcal{S}}$ and a set $\Delta^{\mathcal{S}}$ of **constraints**, i.e., of **labeled diagrams** (P, d) in $G^{\mathcal{S}}$ with $d : \alpha(P) \to G^{\mathcal{S}}$ a morphism in **Base**.

Generalized Sketches

Specification Morphisms

An **morphism** $f : \mathcal{S} \to \mathcal{S}'$ between $\mathcal{N}$-specifications $\mathcal{S} = (G^{\mathcal{S}}, \Delta^{\mathcal{S}})$ and $\mathcal{S}' = (G^{\mathcal{S}'}, \Delta^{\mathcal{S}'})$ is a morphism $f : G^{\mathcal{S}} \to G^{\mathcal{S}'}$ in **Base** preserving constraints, i.e., for all constraints $(P, d) \in \Delta^{\mathcal{S}}$ we have that $(P, d; f) \in \Delta^{\mathcal{S}'}$.



$\mathcal{N}$-specifications and morphisms constitute a category **Ske**($\mathcal{N}$).

Remark: Makkai works with a hierarchy of sketches starting with the base category **Graph**.

Generalized Sketches in view of Institutions

Question

Can we present generalized sketches as institutions?

Answer

Yes! Given a notation $\mathcal{N}$ we get an institution for any semantic universe that allows us to define a corresponding "indexed semantics". In case **Base** has pullbacks, we can define a "fibred semantics" (within **Base**) and get a "pseudo institution" instead.

Signatures and Formulas?

Category of signatures

Given an $\mathcal{N}$-**specification** ($\mathcal{N}$-**sketch**) $\mathcal{S} = (G^{\mathcal{S}}, \Delta^{\mathcal{S}})$ it is quite natural to consider the underlying **Base**-object $G^{\mathcal{S}}$ as a signature and the constraints in $\Delta^{\mathcal{S}}$ as formulas/sentences. So, our **category of signatures** is the category **Base**! (Remember the idea of "signature graphs" in Algebraic Specifications!)

Formulas

Given a "signature", i.e., a **Base**-object G, we denote by

$$Fm(G) = \{(P, d) \mid P \in \Pi, d \in \mathbf{Base}(\alpha(P), G)\}$$

the set of all formulas over G.

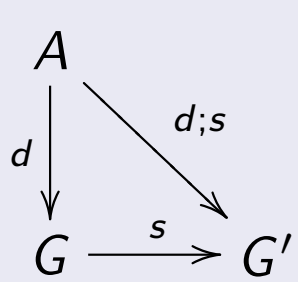
Translation of formulas?

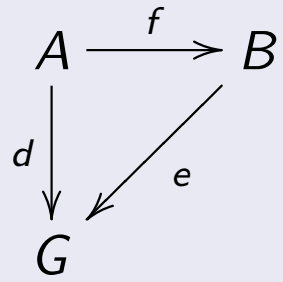
Functors between slices

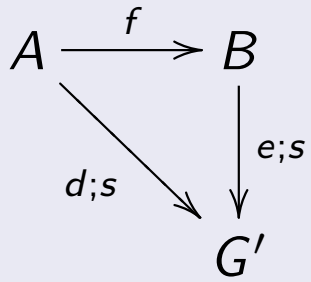
Any morphism $s : G \to G'$ in **Base** defines a functor between slice categories

$$\mathbf{s}_* : (\mathbf{Base} \downarrow G) \to (\mathbf{Base} \downarrow G')$$

by simple post-composition, i.e., by $\mathbf{s}_*(A, d) = (A, d; s)$ for all objects $(A, d : A \to G)$ in $(\mathbf{Base} \downarrow G)$ and by $\mathbf{s}_*(f) = f$ for all morphisms $F : (A, d) \to (B, e)$ in $(\mathbf{Base} \downarrow G)$.



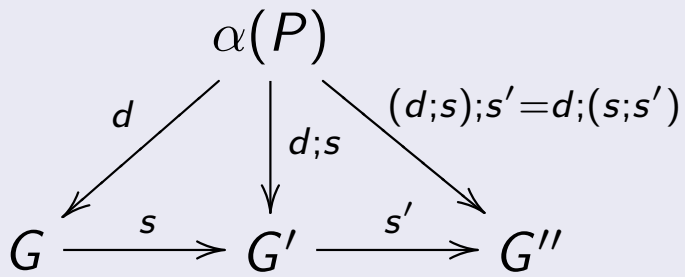




Translation of formulas?

Translation maps and Sentence Functor

Any morphism $s : G \to G'$ in **Base** defines a translation map $Fm(s) : Fm(G) \to Fm(G')$ with $Fm(s)(P, d) = (P, \mathbf{s}_*(d)) = (P, d; s)$.
Since composition in **Base** is associative



the assignments $G \mapsto Fm(G)$ and $s \mapsto Fm(s)$ define a **sentence functor** $Fm : \mathbf{Base} \to \mathbf{Set}$.

Remark*: Actually we consider also "dependencies" between predicates, i.e., Π is a category and $\alpha : \Pi \to \mathbf{Base}$ a functor. In this case, we get a sentences functor $Fm : \mathbf{Base} \to \mathbf{Cat}$.

Indexed Semantics - Informally

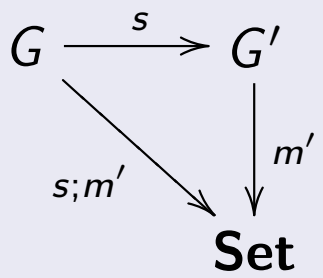
What do I mean by "indexed"?

- For mathematicians/theoreticians it is quite natural to describe the semantics of a formalisms in an "indexed manner", i.e., as an interpretation of syntactic entities in a "semantic universe" **U**.
- "Indexed semantics" is omnipresent in Universal Algebra, Algebraic Specifications, Denotational Semantics, Categorical Logic/Algebra, ..., where **Set** is often the chosen semantic universe.
- "Indexed semantics" supports a clear separation of syntax and semantics, and until 2-3 years ago this was the only way I could think about semantics.

Indexed Semantics - Informally

Indexed semantics for sketches

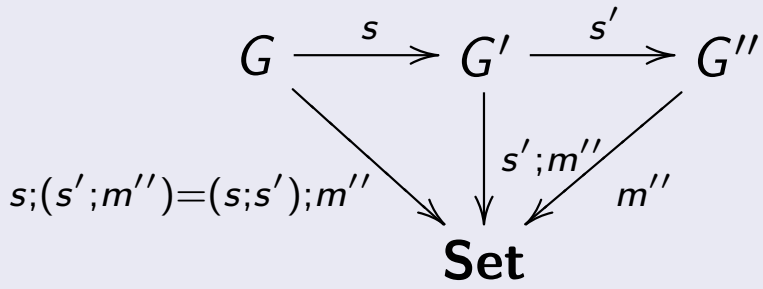
- To define an indexed semantics for sketches we have to chose the base category **Base** and the semantic universe **U** in such a way that the objects in **Base** can be interpreted in **U**.
- If we chose, e.g., **Base** = **Graph** and **U** = **Set** we can define an interpretation (a "model") of a signature (graph) G as a graph homomorphism $m : G \longrightarrow \mathbf{Set}$, i.e., the category of "models" can be given by the "functor category" $mod(G) = [G \longrightarrow \mathbf{Set}]$.
- Any graph homomorphism $s : G \longrightarrow G'$ defines then a functor $mod(s) : mod(G') \longrightarrow mod(G')$ by simple pre-composition, i.e., by $mod(s)(m') = s; m'$ for all "models" $m' : G' \longrightarrow \mathbf{Set}$



Indexed Semantics - Informally

Indexed semantics - model functor

- Again, due to associativity of composition in **GRAPH**, the assignments $G \mapsto mod(G)$ and $s \mapsto mod(s)$ define a "model" functor $mod : \mathbf{Graph}^{op} \longrightarrow \mathbf{Cat}$.



- Also the satisfaction condition can be shown straightforwardly.

Summary/Claim

For any choice of **Base** and **U**, that allows to interpret **Base**-objects in **U** compatible with composition in **Base**, it is relatively straightforward to show that the triple $(\mathbf{Base}, \mathcal{N}, \mathbf{U})$ provides an institution.

Reality in Software Engineering

BUT ! ! !

Reality in Software Engineering

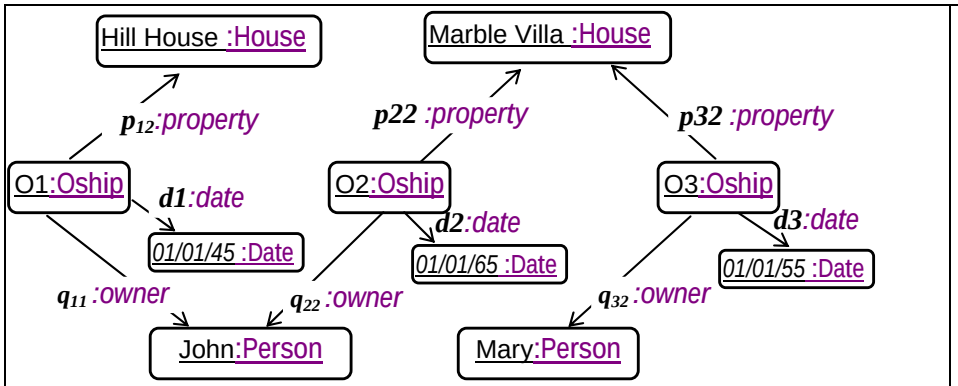
People think differently!

- Software Engineers and especially people in OO-modeling think completely different!
- They think rather in terms of "instances" then of "interpretations (models)"!
- And, even worth, they call a model what we call a specification!

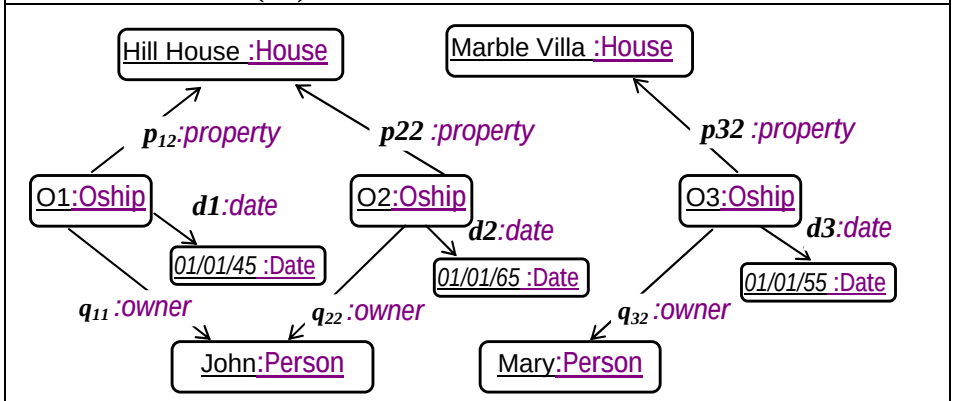
$\Rightarrow$ Note, that this is in accordance with the use of the term "model" in physics, chemistry, and many other areas.

- They iterate the model-instance pattern and call it then metamodel, meta-metamodel, ...

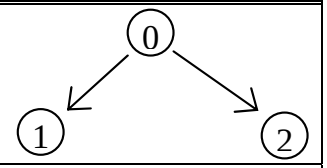
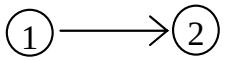
Fibred semantics - Example of instances



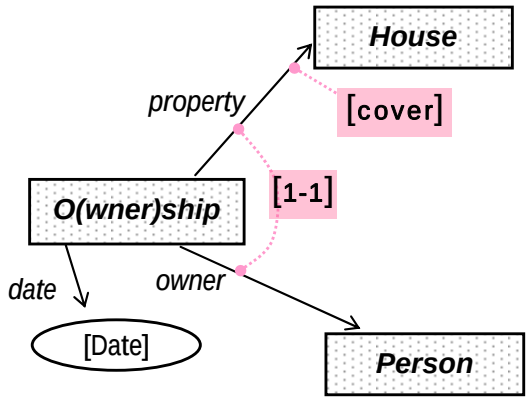
(c1) Valid instance of schema $\mathcal{S}$



(c2) Invalid instance of schema $\mathcal{S}$

Predicate symbol	Arity
[1-1]	
[cover]	

(a) Signature, Π



(b) Schema/ specification/model/sketch, $\mathcal{S}$

Fibred semantics - MOF-based modelling hierarchy

Modelling layers	**Standards and examples**
M_3: Meta-metamodel	MOF
M_2: Metamodel	UML language: Class, Attribute
M_1: Model	A UML model: Class `Person` with Attributes `name` and `address`
M_0: Instance	An instance of `Person`: "Ola Nordmann" living in "Sotraveien 1, Bergen"

Fibred Semantics - Informally

Main ideas and features

- Syntactic entities (models) and semantic entities (instances) live in the same universe.

$\Rightarrow$ This makes it difficult to keep track and to explain to software engineers the difference between syntax and semantics.

- An instance of a syntactic entity (model) is given by a semantic entity and an "abstraction" morphism from the semantic entity into the model.
- It's quite natural to iterate the model-instance pattern.

$\Rightarrow$ This means that the roles "syntax" and "semantics" have only a relative meaning.

$\Rightarrow$ Such a natural iteration is not possible with indexed semantics!

Fibred Semantics - Formally

General Assumption

Let be given a **(diagram predicate) notation** $\mathcal{N} = (\Pi, \alpha)$ over a category **Base** with $\alpha : \Pi \to \mathbf{Base}_0$.
We assume that the base category **Base** has **pullback**.

Semantics of notations

A semantic interpretation of the notation $\mathcal{N}$ is a mapping $[\![..]\!]$, which assigns to each predicate symbol $P \in \Pi$ a set $[\![P]\!] \subset \{\tau \in \mathbf{Base}_1 \mid \mathit{cod}(\tau) = \alpha(P)\}$ of *valid* instances, where $[\![P]\!]$ is assumed to be closed under isomorphisms.

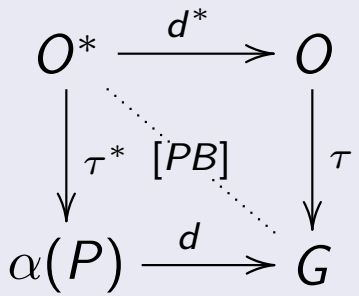
Fibred Semantics - Formally

Semantic of Signatures

For any signature, i.e., any **Base** G the category of **instances** is defined as slice category $\mathbf{Inst}(G) = (\mathbf{Base} \downarrow G)$.

Satisfaction Relation

An instance $\tau : O \to G$ of G **satisfies** a formula (P, d) over G, written $\tau \models (P, d)$, iff $\tau^* = PB_d(\tau) \in [\![P]\!]$.



Fibred Semantics - Formally

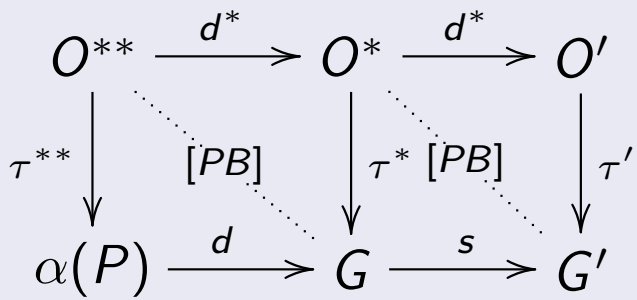
Forgetful Functors by pullbacks

Any morphism $s : G \to G'$ in **Base** induces a functor
$\mathbf{s}^* : (\mathbf{Base} \downarrow G') \to (\mathbf{Base} \downarrow G)$ right-adjoint to the functor
$\mathbf{s}_* : (\mathbf{Base} \downarrow G) \to (\mathbf{Base} \downarrow G')$, where $\mathbf{s}^*(\tau') = PB_s(\tau)$ for any instance $\tau' : O' \to G'$ of G'.

Satisfaction Condition = composition + decomposition of pullbacks

For any morphism $s : G \to G'$, any instance $\tau' : O' \to G'$ of G' and any formula (P, d) over G we have

$$\mathbf{s}^*(\tau') \models_G (P, d) \quad \text{iff} \quad \tau' \models_{G'} (P, d; s).$$



Fibred Semantics - Formally

Why not an institution?

Pullbacks are only determined up to isomorphism. Therefore, for morphisms $s : G \longrightarrow G'$ and $s' : G' \longrightarrow G''$ the functors $(\mathbf{s}; \mathbf{s}')^*$ and $\mathbf{s}^*; \mathbf{s}'^*$ will be not equal but only natural isomorphic.
One can, however, prove that the family of those natural isomorphisms satisfies certain coherence conditions.

Pseudo Functor

The assignments $G \mapsto (\mathbf{Base} \downarrow G)$ and $s \mapsto \mathbf{s}^*$ define a **pseudo functor** $\mathbf{Inst} : \mathbf{Base}^{op} \longrightarrow \mathbf{Cat}$.

Fibred Semantics - Formally

Why the name "fibred"?

For any **indexed category**, i.e., any functor $F : \mathbf{C}^{op} \longrightarrow \mathbf{Cat}$ the Grothendieck construction provides a category $\mathbf{Flat}(\mathbf{F})$ together with a projection functor $pr_F : \mathbf{Flat}(\mathbf{F}) \longrightarrow \mathbf{C}$.
The concept of **(split) fibration** can be seen as an axiomatization of those functors that can be obtained by the Grothendieck construction.

Fibred Semantics vs. Institutions

Summary/claim

Any formalism with fibred semantics will not provide a "model functor" but only a pseudo "model functor".

Questions

- Is the present, well-developed, theory of institutions not applicable to fibred semantics?
- Should we start to extend the theory to pseudo model functors and maybe even pseudo sentence functors?
- Or should we better reconstruct the whole theory in terms of fibrations?

Diagram Predicate Framework (DPF)

DPF - Idea

By Diagram Predicate Framework we name a research activity and an informal research group aimed at to apply generalized sketches in software engineering and to develop further the theory according the practical needs.

DPF - Persons

- Uwe Wolter, Associate Professor, University of Bergen
- Alessandro Rossini, PhD-student, University of Bergen
- Yngve Lamo, Associate Professor, Bergen University College
- Adrian Rutle, PhD-student, Bergen University College
- Zinovy Diskin, Researcher, University of Waterloo, Canada
- Gabriele Taentzer, Professor, University of Marburg, Germany
- ...

Diagram Predicate Framework (DPF)

DPF - Results and applications

- ...
- Generalized sketches as institutions, ACCAT'07
- Context and context awareness, ISOLA'08
- Version control, FASE'09
- Strict meta-modeling, TOOLS'09
- Constraint aware model transformations, NWPT'09 + FASE'10

Diagram Predicate Framework (DPF)

DPF - Current projects, ideas, and open questions

- Distributed version control
- Version control based on categorical complements
- Loose meta-modeling
- Extensions of the framework to typed and/or attributed graphs
- Logic of dependencies
- Specification constraints and their logic
- Generalization of Makkai's construction (pre-sheaf theory)
- Amalgamation in the fibred setting (van-Kampen square)
- Fibred vs. indexed definition of generalized sketches
- Tool design and development
- ...

Thanks

Thanks for your attention!