INDEXED VS. FIBRED STRUCTURES – A FIELD REPORT

UWE WOLTER

Communicated by Dan Timotin

The paper addresses applications of Category Theory in the area of diagrammatic specifications. It outlines corresponding new categorical concepts, constructions and results. We discuss, especially, if and to what extend indexed and fibred structures are appropriate conceptual tools to develop an adequate formalization of diagrammatic specification techniques. The paper reflects the author's experiences and insights while working on a mathematical foundation of Model Driven Software Engineering (MDSE), based on the concept of Generalized Sketch, during the last 14 years.

AMS 2010 Subject Classification: 03G30, 18B25, 18C50, 18D30, 18F20, 68Q55.

Key words: indexed semantics, fibred semantics, Grothendieck construction, amalgamation, van Kampen square.

1. INTRODUCTION AND MOTIVATION

Initially, models were adopted in software development processes for sketching the architectural design or documenting an existing implementation. In contrast, the latest trend in software engineering regards models as first-class entities of the development process. This trend has led to a branch of software engineering, often called Model-Driven Software Engineering (MDSE), which promotes modeling as the main activity of software development and pursues the shift of paradigm from code-centric to model-centric.

In software engineering, a "model" is an abstract representation of certain features and properties of an existing or anticipated software system. In this paper, we use the term "specification", instead of "model", to refer to those abstract representations. Following the tradition in logic and, especially, in "model theory", we use here the term "model" to denote interpretations of specifications in a certain semantic universe.

A wide range of software models are "diagrammatic specifications", i.e., graph-based structures, thus presheaf topoi appear as appropriate categorical structures to build upon a mathematical foundation of MDSE. As a paradigmatic example we consider in this paper the category Graph of (directed multi-) graphs (see Section 4). Classical categorical sketches [1] do not meet the requirements for an appropriate formalization of the wide range of diagrammatic specification techniques we are faced with in MDSE. Therefore we based the development and applications of our Diagram Predicate Framework (DPF) [13, 14, 16, 20, 21, 22] on Generalized Sketches [3, 4, 15].

The original definitions of sketch semantics [4, 5, 15] are based on "indexed semantics in the sense that semantics of sketches is given by interpretations of sketches (specifications) in a semantic universe. In software engineering, however, we are faced with "fibred semantics" where the semantics of specifications is given by "instances" living in the same universe as the specifications [3, 22, 23].

It came as a surprise for us that both paradigms, the "indexed" and the "fibred" one, are by far not equivalent as well from the mathematical as from the application point of view. Especially, the transition from "indexed semantics", omnipresent in mathematics and theoretical computer science, to "fibred semantics" affords surprises.

In the paper we will try to shed some light on the relation between the "indexed" and the "fibred" paradigm. We discuss some advantages and disadvantages of both paradigms especially with respect to applications in MDSE. On the technical side, we focus on model theoretic aspects. In Section 5 we discuss forgetful and free functors, while Section 6 is devoted to amalgamation.

2. INDEXED AND FIBRED STRUCTURES

2.1. Two paradigmatic examples - sets and categories

To begin with, we recapitulate indexed and fibred structures for two paradigmatic concepts in mathematics - sets and categories.

Indexed Sets. Given a set I, an **I-indexed set** $A = (A_i \mid i \in I)$ is a family of sets indexed by the elements in I. An **I-indexed map** $f : A \to B$ between two I-indexed sets is given by an I-indexed family $f = (f_i : A_i \to B_i \mid i \in I)$ of maps. The composition $f;g : A \to C$ of two I-indexed maps $f : A \to B$ and $g : B \to C$ is defined index-wise by the composition of maps $f;g := (f_i;g_i : A_i \to C_i \mid i \in I)$. The identity I-indexed map $id_A : A \to A$ on an I-indexed set A is given by an I-indexed family $id_A := (id_{A_i} \mid i \in I)$ of identity maps. In such a way, I-indexed sets and I-indexed maps give as a category at hand.

More abstractly, we can equivalently describe this category of I-indexed sets as the **functor category** $[\mathsf{P}(I) \to \mathsf{Set}]$ with Set the category of all sets and total maps and $\mathsf{P}(I)$ the discrete category with I as its collection of objects. $\mathsf{P}(I)$ indicates that this discrete category can be seen as the path category generated by the graph with I as set of nodes and the empty set of edges.

Fibred Sets. An I**-fibred set** (A,t) is given by a set A and a map $t : A \to I$. An I-fibred map $g : (A,t) \to (B,u)$ between two I-fibred sets (A,t), (B,u) is given by a map $g : A \to B$ such that $g;u = t$. The composition $f;g : (A,t) \to (C,v)$ of two I-fibred maps $f : (A,t) \to (B,u)$, $g : (B,u) \to (C,v)$ is given by the composition $f;g : A \to C$ of maps in Set. The identity map $id_A : A \to A$ provides obviously the identity I-fibred map $id_{(A,t)} : (A,t) \to (A,t)$ for any I-fibred set (A,t). In such a way, we get a category of I-fibred sets and I-fibred maps. More abstractly, this category is nothing but the **slice category** Set/I (see [1] or [18]).

Indexed Categories. Given a small category I, we consider as I**-indexed categories** arbitrary functors $\mathtt{C} : \mathsf{I} \to \mathsf{Cat}$ with Cat the category of all small categories. That is, I-indexed categories are morphisms in the category CAT. CAT denotes the category with objects all categories like Set, Cat, ..., $[\mathsf{P}(I) \to \mathsf{Set}]$, Set/I, ... and all functors between them. I-indexed functors are natural transformations $\mathtt{C} \Rightarrow \mathtt{D} : \mathsf{I} \to \mathsf{Cat}$ thus we consider the functor category $[\mathsf{I} \to \mathsf{Cat}]$ as our category of I-indexed categories.

Fibred Categories (Fibrations). We follow Barr & Wells [1] and consider an I**-fibred category (fibration)** to be given by a small category E and a functor $\mathtt{P} : \mathsf{E} \to \mathsf{I}$ satisfying the corresponding axioms concerning "cartesian arrows". Our category $\mathsf{Fib}(\mathsf{I})$ of I-fibred categories is the full subcategory of the slice category Cat/I with objects all I-fibred categories.

2.2. Indexed or fibred – An informal discussion

Whenever we want or need to define structures, we do have the choice between the "indexed" and the "fibred" way.

Signatures. Algebraic signatures, for example, can be defined in the "indexed way": A signature $\Sigma = (S, OP)$ is given by a set S of sort symbols and an $(S^* \times S)$- indexed set $OP = (OP_{w,s} \mid w \in S^*, s \in S)$ of operation symbols. Or, we can chose the "fibred way": A signature $\Sigma = (S, OP, \alpha)$ is given by a set S of sort symbols, a set OP of operation symbols and an arity function $\alpha : OP \to (S^* \times S)$. The "indexed way" is not our preferred choice. Usually, we do have only finite many operation symbols thus infinite many of the sets $OP_{w,s}$ will be just empty. Moreover, the "indexed way" allows "overloading" of operation symbols since the sets $OP_{w,s}$ are not assumed to be disjoint. If we want to ensure that all Σ-terms do have a unique sort, we have to resolve this overloading when defining Σ-terms! The carriers of Σ-algebras, however, are usually defined in an "indexed way", namely as S-indexed sets.

Indexed mindset. A high percentage of mathematicians and theoretical computer scientists are growing up in an "indexed mindset". We learn that syntax and semantics should be strictly separated. Syntactic entities live on one side and are interpreted in a chosen "semantic universe" on the other side, like the category Set of sets and total maps , Par of sets and partial maps, Rel of sets and binary relations or PO of partial orders, for example. We call this the **indexed semantics** or **semantics-as-interpretation** paradigm. There is a huge tower of theory build upon this paradigm. As areas, relying on this paradigm, we mention only a few: Universal Algebra, Algebraic Specifications, Model Theory, Functorial Semantics, Denotational Semantics, Categorical Algebra. Also the concept of Institution [2, 8] reflects to a great extend the semantics-as-interpretation paradigm (compare Section 5).

Software engineering. There are, at least, two reasons why indexed semantics may be not fully adequate to formalize certain concepts, structures and constructions in software engineering. First, it doesn't reflect the "model-instance pattern" omnipresent in software engineering. An Object Diagram, for example, is an instance of a Class Diagram in the sense that there is a graph homomorphism from the underlying graph of the Object Diagram into the underlying graph of the Class Diagram, i.e., the Object Diagram is typed (fibred) over the Class Diagram. We call this kind of fibred approach to semantics the **fibred semantics** or **semantics-as-instance** paradigm. Second, indexed semantics is a rigid two-level approach with only one abstraction level – syntax. In software engineering, as in nearly any area in science, life and industry, we are, however, faced with arbitrary deep hierarchies of abstractions. In contrast to the semantics-as-interpretation pattern, the semantics-as-instance pattern can be neatly iterated and offers an appropriate formalization of arbitrary deep hierarchies of abstractions [14, 21, 22].

This observation triggered, more than a decade ago, our decision to develop a fibred semantics for Generalized Sketches [4, 5, 15], and we made some progress in this direction [3, 21, 22, 23, 24, 25]. Our initial idea to achieve the shift of paradigm by a simple translation of constructions and results from the indexed setting to the fibred one, was quite naive as we will see.

3. FROM INDEXED TO FIBRED SEMANTICS

In this section we outline and discuss transitions between indexed and fibred semantics.

3.1. Two paradigmatic examples - sets and categories

Indexed vs. Fibred Sets. For each I-indexed set $A : \mathsf{P}(I) \to \mathsf{Set}$ the disjoint union construction provides an I-fibred set $(Gr(A), pr_A)$,where

Gr stands for "Grothendieck construction" (see the next paragraph), with $Gr(A) := \{\langle i, a\rangle \mid i \in I, a \in A(i)\}$ and $pr_A : Gr(A) \to I$ given by $pr_A(\langle i, a\rangle) := i$. The assignments $A \mapsto (Gr(A), pr_A)$ extend to a functor $\mathtt{Gr}_I : [\mathsf{P}(I) \to \mathsf{Set}] \to \mathsf{Set}/I$.

The other way around, the formation of pre-images (fibers) provides for any I-fibred set (B, t) an I-indexed set $Fb(B, t) : \mathsf{P}(I) \to \mathsf{Set}$ with $Fb(B, t)(i) := t^{-1}(i) = \{b \in B \mid t(b) = i\}$. The assignments $(B, t) \mapsto Fb(B, t)$ extend to a functor $\mathtt{Fb}_I : \mathsf{Set}/I \to [\mathsf{P}(I) \to \mathsf{Set}]$. For any set I the functors $\mathtt{Gr}_I$ and $\mathtt{Fb}_I$ establish an equivalence between the categories $[\mathsf{P}(I) \to \mathsf{Set}]$ and Set/I.

Indexed vs. Fibred Categories. For categories the equivalence breaks apart. The generalization of the disjoint union construction is called **Grothendieck construction** [1, 17] and transforms any I-indexed category $\mathtt{C} : \mathsf{I} \to \mathsf{Cat}$ into an I-fibred category $pr_{\mathsf{I}} : Gr(\mathsf{I}) \to \mathsf{I}$. pr_{I} is a **split fibration** [1] and the generalization of the fiber-construction gives us only an equivalence between $[\mathsf{I} \to \mathsf{Cat}]$ and the full subcategory of Cat/I of all split fibrations. For arbitrary fibrations the fiber-construction provides, in general, only pseudo-functors and pseudo-natural transformations thus the corresponding Grothendieck construction gives us only "strict 2-equivalences of 2-categories" at hand. Both kinds of equivalences are not of much help in our applications (see Section 4).

3.2. A more general picture

We don't distinguish in this paper between signatures and specifications. We just assume that we do have a **category** Spec **of specifications**.

Indexed Semantics. To define an indexed semantics for specifications, we choose, first, a **semantic universe** SU. To interpret specifications, we have to find, second, a **meta-universe** MU such that there exists an embedding functor $\mathtt{in} : \mathsf{Spec} \to \mathsf{MU}$ and an object $\mathtt{ex}(\mathsf{SU})$ extracted from the semantic universe SU. As **models** of a **specification** Sp we choose all or only certain morphisms $m : \mathtt{in}(Sp) \to \mathtt{ex}(\mathsf{SU})$ in MU. Third, we need a mechanism, relying on MU, SU and the extraction process for $\mathtt{ex}(\mathsf{SU})$, allowing us to turn for each specification Sp the collection of chosen morphisms $m : \mathtt{in}(Sp) \to \mathtt{ex}(\mathsf{SU})$ into a category $\mathtt{Mod}(Sp)$ of all models of Sp in SU.

In case of indexed sets, we do have $\mathsf{Spec} = \mathsf{Set}$, $\mathsf{MU} = \mathsf{CAT}$ and a functor $\mathtt{in} : \mathsf{Set} \to \mathsf{CAT}$ assigning to each set I the corresponding discrete category $\mathtt{P}(I)$ in CAT. Further, we have $\mathtt{ex}(\mathsf{SU}) = \mathsf{SU} = \mathsf{Set}$ and use the mechanism "functor category" to define model categories $\mathtt{Mod}(I) := [\mathtt{P}(I) \to \mathsf{Set}]$.

In case of indexed categories, we chose $\mathsf{Spec} = \mathsf{Cat}$, $\mathsf{MU} = \mathsf{CAT}$ and a functor $\mathtt{in} : \mathsf{Cat} \to \mathsf{CAT}$ embedding the category of small categories into CAT.[1] Further, we chose $\mathtt{ex}(\mathsf{SU}) = \mathsf{SU} = \mathsf{Cat}$ and used again the mechanism "functor category" to define model categories $\mathtt{Mod}(\mathsf{C}) := [\mathtt{in}(\mathsf{C}) \to \mathsf{Cat}]$.

Fibred semantics. To define a fibred semantics for specifications, we have to choose, first, a **common universe** CU together with an embedding functor $\mathtt{em} : \mathsf{Spec} \to \mathsf{CU}$. As category $\mathtt{Inst}(Sp)$ of all **instances** of a specification Sp we choose, second, a subcategory of the slice category $\mathsf{CU}/\mathtt{em}(Sp)$.

In case of fibred-sets, we have $\mathsf{Spec} = \mathsf{CU} = \mathsf{Set}$ and $\mathtt{em} = id_{\mathsf{Set}}$, while $\mathtt{Inst}(I) := \mathsf{Set}/I$ for any set I. For indexed categories, we chose $\mathsf{Spec} = \mathsf{CU} = \mathsf{Cat}$ and $\mathtt{em} = id_{\mathsf{Cat}}$, while $\mathtt{Inst}(\mathsf{I})$ is chosen as the full subcategory of Cat/I with objects all I-fibred categories.[2]

Grothendieck construction. We abstract from the two examples in Subsection 3.1. Transforming an indexed semantics into a fibred semantics means that we have a construction transforming any model $m : \mathtt{in}(Sp) \to \mathtt{ex}(\mathsf{SU})$ into an instance of Sp, i.e., into an object $Gr(m)$ together with a morphism $pr_m : Gr(m) \to \mathtt{em}(Sp)$. Moreover, this construction should extend to morphisms, in such way, that we get for each specification Sp a functor from $\mathtt{Mod}(Sp)$ into $\mathtt{Inst}(Sp)$. We will keep the name "Grothendieck" for those constructions.

4. INDEXED AND FIBRED SEMANTICS FOR GRAPHS

As a non-trivial, but still simple, example of "diagrammatic specifications" we consider as our category Spec of specifications the category Graph of small (directed multi) graphs. A (small) **graph** $G = (G_V, G_E, sc^G, tg^G)$ is given by a collection (set) G_V of nodes, a collection (set) G_E of edges and two maps $sc^G, tg^G : G_E \to G_V$. A morphism $\varphi = (\varphi_V, \varphi_E) : G \to K$ between graphs is given by two maps $\varphi_V : G_V \to K_V$, $\varphi_E : G_E \to K_E$ such that $sc^G;\varphi_V = \varphi_E;sc^K$ and $tg^G;\varphi_V = \varphi_E;tg^K$. Composition of graph morphisms is defined by componentwise composition of maps. Note, that the category Graph of small graphs is isomorphic to the presheaf topos $[\mathsf{MG} \to \mathsf{Set}]$ with MG ("Metamodel of Graphs") the category: $id_E \circlearrowright E \underset{tg}{\overset{sc}{\rightrightarrows}} V \circlearrowleft id_V$.

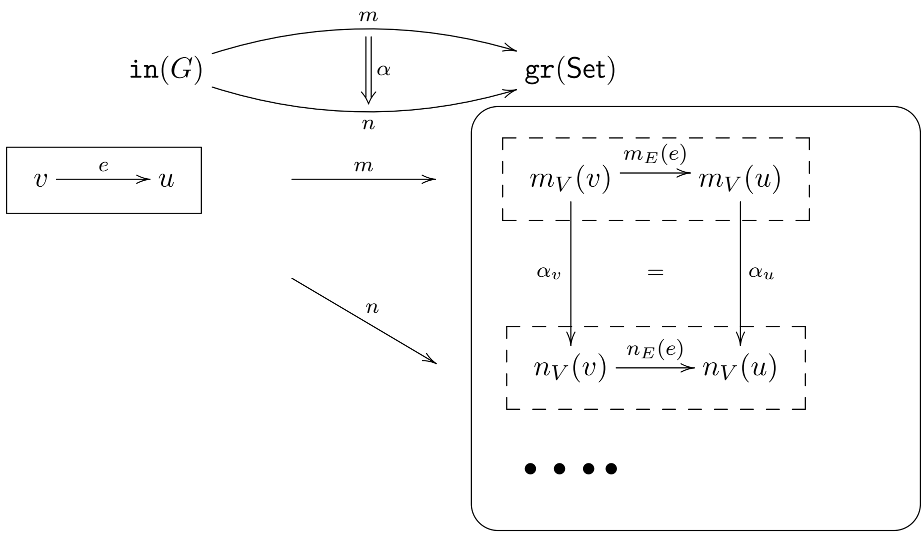
By GRAPH we denote the category containing as objects, especially, the underlying graphs of categories like Set, Cat, Graph, Par, Rel, ..., $[\mathtt{P}(I) \to \mathsf{Set}]$,

[1] We consider an object C in Cat and its counterpart $\mathtt{in}(\mathsf{C})$ in CAT as distinct but isomorphic mathematical entities! The isomorphisms between C and $\mathtt{in}(\mathsf{C})$ live in the same category as CAT and the functor $\mathtt{in}$. We consider this category as a "regulative idea" ([18],p. 5).

[2] Examples for non-trivial cases "$\mathtt{ex}(\mathsf{SU})$" and "$\mathtt{em}$" are discussed in Section 4.

$\mathsf{Set}/I, \ldots$.

Indexed semantics. Often, nodes in diagrams in software engineering are interpreted as sets while edges represent maps. Therefore, we may choose $\mathsf{SU} := \mathsf{Set}$. Further, we choose $\mathsf{MU} := \mathsf{GRAPH}$, $\mathtt{in} : \mathsf{Graph} \to \mathsf{GRAPH}$ the embedding of Graph into GRAPH and $\mathtt{ex}(\mathsf{Set}) := \mathtt{gr}(\mathsf{Set})$ the underlying graph of the category Set.[3] As collection of models of a small graph G we can choose then the whole hom-set $\mathsf{GRAPH}(\mathtt{in}(G), \mathtt{gr}(\mathsf{Set}))$. Fortunately, we can borrow the composition in Set to define morphisms between those models: A **natural transformation** $\alpha : m \Rightarrow n$ between two models (interpretations) $m, n : \mathtt{in}(G) \to \mathtt{gr}(\mathsf{Set})$ is given by a family $\{\alpha_v : m_V(v) \to n_V(v) \mid v \in \mathtt{in}(G)_V\}$ of morphisms in Set such that $m_E(e); \alpha_u = \alpha_v; n_E(e)$ for all edges $e : v \to u$ in $\mathtt{in}(G)$.



In the usual way, natural transformations between interpretations define a category on the hom-set $\mathsf{GRAPH}(\mathtt{in}(G), \mathtt{gr}(\mathsf{Set}))$. We call those categories **interpretation categories** and denote them by $[\mathtt{in}(G) \to \mathsf{Set}]$ (see [27]). In other words: We use the mechanism "interpretation category" to define our categories of models $\mathtt{Mod}(G) := [\mathtt{in}(G) \to \mathsf{Set}]$.[4]

There are cases where edges in a diagram represent partial maps or binary relations. Then we choose simply $\mathsf{SU} := \mathsf{Par}$ or $\mathsf{SU} := \mathsf{Rel}$, respectively, instead of $\mathsf{SU} := \mathsf{Set}$. In other cases we may interpret nodes as graphs and edges as graph morphisms thus we have to choose $\mathsf{SU} := \mathsf{Graph}$.

[3] The notation $\mathtt{gr}$ for the underlying graph of a category should not be confounded with Gr, which stands for the Grothendieck construction.

[4] Barr & Wells [1] use the same mechanism to define categories of models of sketches only that they don't coin explicitly the concept "interpretation category".

Interpretation vs. functor categories. Of course, we could also use the mechanism "functor category" to define indexed semantics in an equivalent way. We choose $\mathsf{MU} := \mathsf{CAT}$ and $\mathtt{in} : \mathsf{Graph} \to \mathsf{CAT}$ assigns to each small graph G the corresponding path category $\mathsf{P}(G)$. Categories of models are then functor categories $[\mathsf{P}(G) \to \mathsf{SU}]$. This is, however, not the appropriate way to propagate applications of category theory in software engineering! We can not tell a software engineer first you have to transform your diagram into a category before you can make a meaning out of it. Moreover, a finite diagram G may generate an infinite path category $\mathsf{P}(G)$ and infinite structures can not be handled by a computer. A computer can only handle finite representations of infinite structures!

Fibred semantics. A fibred semantics is simply obtained by choosing $\mathsf{CU} := \mathsf{Graph}$, $em := id_{\mathsf{Graph}}$ and $\mathtt{Inst}(G) := \mathsf{Graph}/G$ for any small graph G.

Grothendieck construction. For any model $m : \mathtt{in}(G) \to \mathtt{gr}(\mathsf{Set})$ we construct a small graph $Gr(m)$ as follows:

- **nodes:** $\langle v, x\rangle$ with v a node in G and $x \in m_V(v)$
- **arrows:** $\langle e, x\rangle : \langle v, x\rangle \to \langle u, y\rangle$ with $e : v \to u$ an edge in G and $y = m_E(e)(x)$.

We obtain a graph morphism $pr_m : Gr(m) \to G$ with $pr_m(\langle v, x\rangle) := v$ for any node $\langle v, x\rangle$ in $Gr(m)$ and $pr_m(\langle e, x\rangle) := e$ for any arrow $\langle e, x\rangle$ in $Gr(m)$. This construction extends to model morphisms and we get a functor from $\mathtt{Mod}(G) = [\mathtt{in}(G) \to \mathsf{Set}]$ into $\mathtt{Inst}(G) = \mathsf{Graph}/G$ for any small graph G.

The outlined Grothendieck construction generalizes straightforwardly to the cases $\mathsf{SU} = \mathsf{Par}$ or $\mathsf{SU} = \mathsf{Rel}$ where we get functors from $[\mathtt{in}(G) \to \mathsf{Par}]$ or $[\mathtt{in}(G) \to \mathsf{Rel}]$, respectively, into Graph/G. In other word, our choice to take as fibred semantics the whole slice category $\mathtt{Inst}(G) = \mathsf{Graph}/G$ means that we include also the cases $\mathsf{SU} = \mathsf{Par}$ and $\mathsf{SU} = \mathsf{Rel}$ on the indexed side.

In case $\mathsf{SU} = \mathsf{Graph}$, however, the Grothendieck construction doesn't provide a functor into Graph/G but into $\mathsf{Graph}/R(G)$ where $R(G)$ is the reflexive graph generated by G [23]. So, in this case we have to vary the fibred semantics by choosing a non-trivial functor $\mathtt{em} : \mathsf{Graph} \to \mathsf{Graph}$ with $\mathtt{em}(G) := R(G)$.

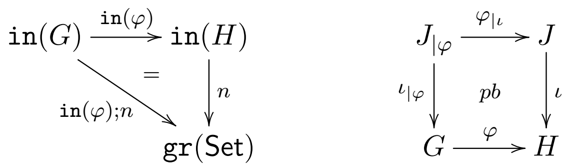
5. FORGETFUL AND FREE FUNCTORS FOR GRAPHS

Fibred semantics gives us a neat formalization of metamodelling at hand but doesn't behave as nice as indexed semantics when it comes to important "model theoretic" constructions. In this section we discuss forgetful and free functors for indexed and fibred semantics of graphs.

5.1. **Forgetful functors**

Forgetful functors are inherent to indexed semantics while it is more technically involved to define forgetful functors for fibred semantics.

Indexed semantics. Forgetful functors are simply provided by pre-composition in GRAPH. Any graph homomorphism $\varphi : G \to H$ induces a **forgetful functor** $\mathtt{Mod}(\varphi) : \mathtt{Mod}(H) \to \mathtt{Mod}(G)$ with $\mathtt{Mod}(\varphi)(n) := \mathtt{in}(\varphi); n$ for any H-model $n : \mathtt{in}(H) \to \mathtt{gr}(\mathsf{Set})$. Note, that natural transformations between interpretations are pre-composed with a graph morphism in the same way as natural transformations between functors are pre-composed with a functor. Since composition in GRAPH is associative, the assignments $G \mapsto \mathtt{Mod}(G)$ and $\varphi \mapsto \mathtt{Mod}(\varphi)$ define a **(model) functor** $\mathtt{Mod} : \mathsf{Graph}^{op} \to \mathsf{CAT}$.



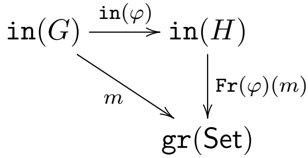
Fibred semantics. First, we have to decide for an arbitrary but fixed choice of pullbacks in Graph. Then, any graph morphism $\varphi : G \to H$ induces a **forgetful functor** $Inst(\varphi) : \mathtt{Inst}(H) \to \mathtt{Inst}(G)$ with $\mathtt{Inst}(\varphi)(\iota) := (J_{|\varphi}, \iota_{|\varphi})$ for any instance $\iota : J \to H$ of H, where $(J_{|\varphi}, \iota_{|\varphi})$ is given by the chosen pullback of ι along φ (see the right-hand diagram above). The construction of chosen pullbacks is, in general, only compositional "up to isomorphism" thus the assignments $G \mapsto \mathtt{Inst}(G)$ and $\varphi \mapsto \mathtt{Inst}(\varphi)$ will, in general, only define a **pseudo (instance) functor** $\mathtt{Inst} : \mathsf{Graph}^{op} \to \mathsf{CAT}$ [3, 17].

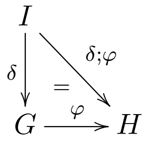
5.2. **Free functors**

Concerning free functors, we face exactly the opposite picture. Free functors are inherent to fibred semantics while it is more technically involved to define free functors for indexed semantics.

Fibred semantics. Any graph homomorphism $\varphi : G \to H$ induces by simple post-composition, a **(change-of-base) functor** $\mathtt{Ch}(\varphi) : \mathtt{Inst}(G) \to \mathtt{Inst}(H)$, with $\mathtt{Ch}(\varphi)(I, \delta) := (I, \delta; \varphi)$ for any G-instance (I, δ). $\mathtt{Ch}(\varphi)$ is left-adjoint to $\mathtt{Inst}(\varphi) : \mathtt{Inst}(H) \to \mathtt{Inst}(G)$. The assignments $G \mapsto \mathtt{Inst}(G)$ and

$\varphi \mapsto \mathtt{Ch}(\varphi)$ define a functor $\mathtt{Ch} : \mathsf{Graph} \to \mathsf{CAT}$.





Indexed semantics. To obtain for any graph homomorphism $\varphi : G \to H$ a **free functor** $\mathtt{Fr}(\varphi) : \mathtt{Mod}(G) \to \mathtt{Mod}(H)$, i.e., a functor left-adjoint to the forgetful functor $\mathtt{Mod}(\varphi) : \mathtt{Mod}(H) \to \mathtt{Mod}(G)$, we have to construct for any G-model $m : \mathtt{in}(G) \to \mathtt{gr}(\mathsf{Set})$ an H-model $\mathtt{Fr}(\varphi)(m) : \mathtt{in}(H) \to \mathtt{gr}(\mathsf{Set})$ together with a natural transformation $\eta_m : m \Rightarrow \mathtt{in}(\varphi); \mathtt{Fr}(\varphi)(m)$ satisfying the corresponding universal property. If we interpret a graph G as an algebraic signature declaring a set G_V of sorts and a unary operation $e : v \to u$ for each edge in G_E, we can describe $\mathtt{Fr}(\varphi)(m)$ as the "H-algebra freely generated by the G-algebra m along the signature morphism φ". The construction of "free algebras" is only compositional "up to isomorphism" thus the assignments $G \mapsto \mathtt{Mod}(G)$ and $\varphi \mapsto \mathtt{Fr}(\varphi)$ define, in general, only a pseudo functor $\mathtt{Fr} : \mathsf{Graph} \to \mathsf{CAT}$.

Be aware, that the "free algebra construction" is not fully reflecting what we do in the fibred setting. In the fibred setting, we have $(\delta; \varphi)^{-1}(v) = \emptyset$ for all nodes $v \in H_V \setminus \varphi_V(G_V)$ while $\mathtt{Fr}(\varphi)(m)_V(v)$ will be not empty if there is a node $u \in G_V$ with $m_V(u) \neq \emptyset$ and a chain of edges in H from $\varphi_V(u)$ to v. To mimic the effect of change-of-base functors, we can choose, e.g., $\mathsf{SU} := \mathsf{Par}$ since the construction of "free partial algebras along a signature morphism" doesn't introduce any new definedness for partial operations [19, 26].

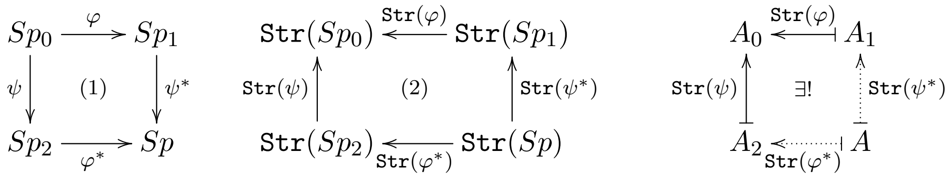
6. AMALGAMATION FOR GRAPHS

6.1. Compositionality in general

We consider an arbitrary category Spec of specifications and specification morphisms together with a "semantic" functor $\mathtt{Str} : \mathsf{Spec}^{op} \to \mathsf{CAT}$ assigning to each specification Sp a category $\mathtt{Str}(Sp)$ of structures, like interpretations or instances, for example, complying with Sp and assigning to each specification morphism $\varphi : Sp_1 \to Sp_2$ a (forgetful) functor $\mathtt{Str}(\varphi) : \mathtt{Str}(Sp_2) \to \mathtt{Str}(Sp_1)$.

Compositionality is an important and well-known concept in theoretical computer science [7]. It is a method to uniquely and correctly compose (overlapping) semantics of components of an already composed specification. The composition of specifications is usually carried out with the help of colimits, i.e., the category Spec is assumed to be finitely cocomplete. E.g. in the left diagram in the figure below specifications Sp_1 and Sp_2 are related via the common part Sp_0 whose role as subspecification of Sp_1 and Sp_2 is formalized with specification morphisms φ and ψ, resp. Syntactic composition is carried out by constructing the pushout of φ and ψ.

Compositionality means that the "semantic" functor $\mathtt{Str}$ is continuous, i.e., transforms colimits in Spec into limits in CAT. Especially, the pushout (1) of specifications should be transformed into a pullback (2) of categories.

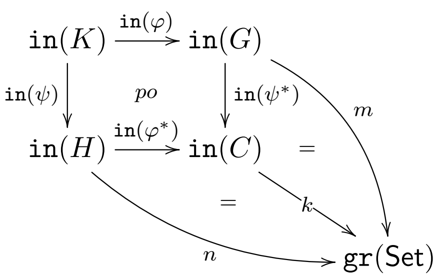


To achieve compositionality for pushouts, we need **amalgamation** of structures: For any structures A_0, A_1, and A_2 complying to Sp_0, Sp_1, and Sp_2 resp., which are related to each other according to the action of the functor $\mathtt{Str}$, i.e., $\mathtt{Str}(\varphi)(A_1) = A_0 = \mathtt{Str}(\psi)(A_2)$, there has to exist a unique structure A, complying to Sp, such that $\mathtt{Str}(\psi^*)(A) = A_1$ and $\mathtt{Str}(\varphi^*)(A) = A_2$.

The most frustrating surprise in our project "fibred semantics for Generalized sketches" was that amalgamation of instances turned out to be quite complicated while amalgamation of models is nearly trivial. In the remaining part of this section, we discuss this issue in more detail and outline some new results.

6.2. **Model amalgamation**

For our sample formalism with $\mathsf{Spec} = \mathsf{Graph}$ and $\mathtt{Str}$ the model functor $\mathsf{Mod} : \mathsf{Graph}^{op} \to \mathsf{CAT}$, defined in Subsection 5.1, we get **amalgamation of models** for free. The embedding $\mathtt{in} : \mathsf{Graph} \to \mathsf{GRAPH}$ preserves pushouts thus we get for arbitrary pushouts in Graph:



There exists for any **coherent pair of models**, i.e., for any $m : \mathtt{in}(G) \to \mathsf{gr}(\mathsf{Set})$ and $n : \mathtt{in}(H) \to \mathsf{gr}(\mathsf{Set})$ with $\mathsf{Mod}(\varphi)(m) = \mathtt{in}(\varphi); m = \mathtt{in}(\psi); n = \mathsf{Mod}(\psi)(n)$, a unique model $k : \mathtt{in}(C) \to \mathsf{gr}(\mathsf{Set})$ such that $\mathsf{Mod}(\varphi^*)(k) = \mathtt{in}(\varphi^*); k = n$ and $\mathsf{Mod}(\psi^*)(k) = \mathtt{in}(\psi^*); k = m$.

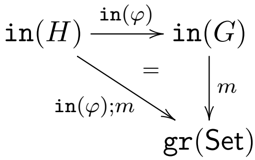
6.3. Model amalgamation in view of the Grothendieck construction

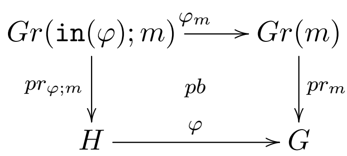
To find out, what amalgamation of instances could be, we analyze the translation of model amalgamation into the fibred setting by means of the Grothendieck construction.

Model reduction into pullback. For any graph morphism $\varphi : H \to G$ and any interpretation $m : \mathtt{in}(G) \to \mathtt{gr}(\mathsf{Set})$ the assignments

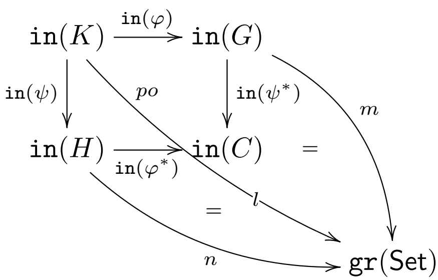
- $\varphi_{m,V}(\langle v, x\rangle) = \langle \varphi_V(v), x\rangle$ for any node $\langle v, x\rangle$ in $Gr(\mathtt{in}(\varphi); m)$ and
- $\varphi_{m,E}(\langle e, x\rangle) = \langle \varphi_E(e), x\rangle$ for any arrow $\langle e, x\rangle$ in $Gr(\mathtt{in}(\varphi); m)$

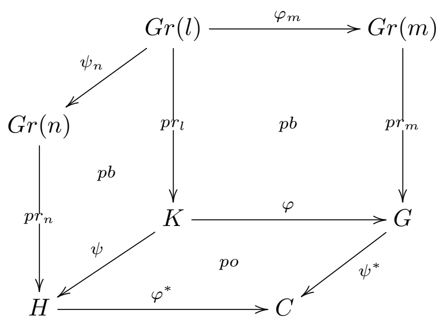
define a graph morphism $\varphi_m : Gr(\mathtt{in}(\varphi), m) \to Gr(m)$ such that the following right diagram is a pullback diagram in Graph (compare ex. 1.10.4 in [10])



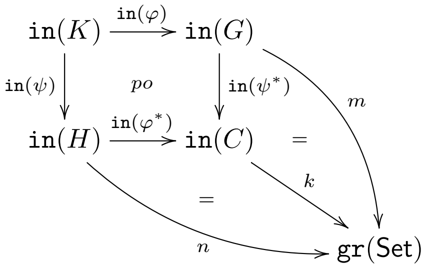


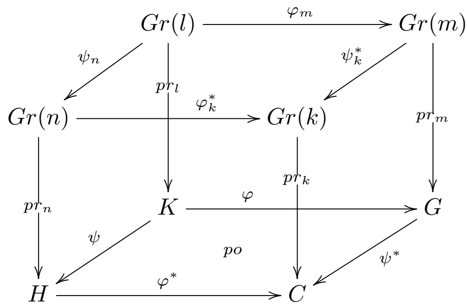
Coherent models into pullback-pushout half cube. In such a way, the Grothendieck construction transforms any coherent pairs $m : \mathtt{in}(G) \to \mathtt{gr}(\mathsf{Set})$, $n : \mathtt{in}(H) \to \mathtt{gr}(\mathsf{Set})$ of models into a **pullback-pushout half cube** where $l := \mathtt{in}(\varphi); m = \mathtt{in}(\psi); n$.





Mediator into pullback completion. Finally, the existence of a unique mediating morphism k is turned into the existence of a pullback completion of the pullback-pushout half cube. That is, we get a commutative cube where also the front and the right faces are pullbacks. Note an important difference between the indexed and the fibred setting: While the mediator is unique "on the nose" a corresponding pullback completion will be only unique "up to isomorphism".





6.4. **Instance amalgamation**

Instance amalgamation. Summarizing our analysis in Subsection 6.3, we can conclude that instance amalgamation means to construct pullback completions for pullback-pushout half cubes in the common universe CU, i.e., Graph in our case. We have no idea how to do this in arbitrary categories CU. If CU is a topos, however, we know that for a pullback completion of a pullback-pushout half cube the resulting top square becomes also a pushout ([9] 15.3). So, in topoi, the only chance to get a pullback completion is to construct a pushout of the span of morphisms on top of the cube and to hope that the resulting commutative front and right faces are pullbacks. We consider this as the most reasonable procedure to construct amalgamation of instances.

Counterexample. Also for the cases SU = Par or SU = Rel we do have trivially model amalgamation for arbitrary pushouts in Spec = Graph and the corresponding Grothendieck constructions transform also in these cases coherent pairs of models into pullback-pushout half cubes in CU = Graph that do have a pullback completion. There are, however, pullback-pushout half cubes that are not obtained by any variant of the Grothendieck construction and thus may not have pullback completions. We consider the two examples in Fig. 1 of pullback-pushout half cubes in Set. The pushout of the span of maps on the top of the left example produces a set with two elements and it is easy to check that the resulting front and right faces are pullbacks. Intertwining the two equivalences in I, obtained as the kernels of the two maps a‘ and r‘, respectively, gives us the right example. On the one hand, pullback complements for the right and the front face with sets over S containing two elements will always yield a non-commutative top face. On the other hand, the pushout on the top face creates a singleton set and the resulting front and left squares are not pullbacks.

Amalgamable instances. After facing the hard fact that amalgamation of coherent instances will be not possible in very many cases, while amalgama-

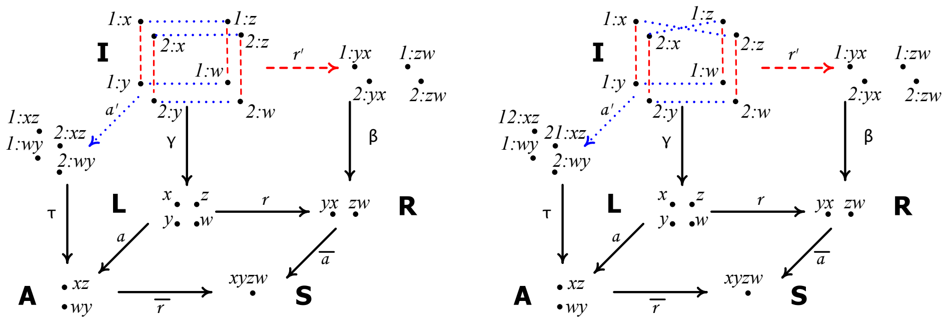


Figure 1 – Existence and non-existence of pullback completions.

tion of coherent models is always possible, two questions arise naturally. First, we may ask for a characterization of those pairs of coherent instances that can be amalgamated. Or, in other words: What pullback-pushout half cubes in CU do have a pullback completion?

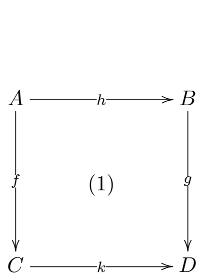
We are not able to give an answer for arbitrary categories or arbitrary topoi CU. However, for arbitrary presheaf topoi, i.e., for functor categories $\mathsf{CU} = [\mathsf{C} \to \mathsf{Set}]$ with C a small category, a necessary and sufficient characterization is given in [25]. Intuitively, this characterization says that a pullback-pushout half cube does have a pullback completion if, and only if, the top span of morphisms is a multiple copy of the bottom span of morphisms (compare the left example in Fig. 1). In other words: There are no effects on the semantic level that are not reflected on the syntax level.

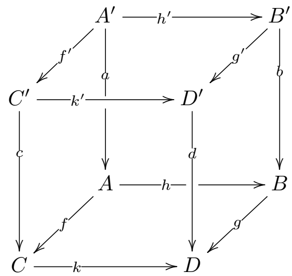
6.5. **Van Kampen squares and van Kampen colimits**

Second, it is important to know for what bottom pushouts in CU **all** (!) pullback-pushout half cubes do have a pullback completion? More intuitively: For what bottom spans of morphisms the intertwining of equivalences, we have seen in the counter example in Fig. 1, is not possible? It was a kind of bitter surprise for the author that we ended up, in such a way, with the question: What are the van Kampen squares in CU?

Van Kampen squares. A pushout (1) is a **van Kampen square** if, for any commutative cube (2) with (1) in the bottom and where the back and the left faces are pullbacks, the following equivalence holds:

The top face is pushout iff the front and right faces are pullbacks:





(2)

Adhesive categories. A sufficient condition for van Kampen squares in arbitrary topoi is that one of the morphisms f or h in the square above is monic. This observation gave rise to the concept of Adhesive Category coined by Lack and Sobociński [12] and used, e.g., to systematize and generalize essential concepts, constructions and results in the area of Graph transformations [6].

Unique path condition. In practice, we meet, however, situations where none of the two morphisms in the span is monic, but we still need amalgamation of all coherent pairs of instances.

In [25] we give a sufficient and necessary characterization of van Kampen squares in arbitrary presheaf topoi by means of a "**unique path condition**". Referring to our counter example (see Fig. 2), the unique path condition means informally that the kernel of a and the kernel of r should not interact in such a way that there are two essentially different ways to find out that two items in L have to be identified by $a;\overline{r} = r;\overline{a}$. In the counter example we have, e.g., the two distinct alternating paths $x \xrightarrow{a} xz \xleftarrow{a} z \xrightarrow{r} zw \xleftarrow{r} w \xrightarrow{a} wy \xleftarrow{a} y$ and $x \xrightarrow{r} yx \xleftarrow{r} y$ identifying x and y. Two variations of the example, where

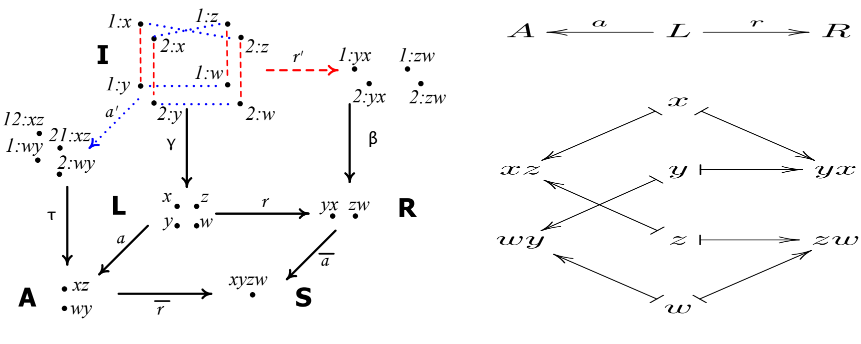


Figure 2 – Example – unique path condition not satisfied.

the unique path condition is satisfied while still producing a singleton set as pushout object on the bottom, are pictured in Fig. 3.

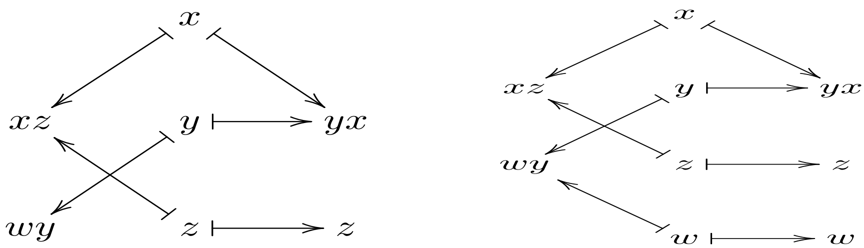


Figure 3 – Example – unique path condition satisfied.

Van Kampen colimits We asked then if and how the results in [25] could be generalized. We introduce in [11] the concept of van Kampen colimit and give a sufficient and necessary characterization of van Kampen colimits in arbitrary presheaf topoi by means of a generalized unique path condition.

7. CONCLUDING REMARKS

What did we learned during our journey from traditional indexed to fibred semantics and from traditional specifications to diagrammatic specifications? What insights and recommendations we tried to communicate in this paper?

First, we shouldn't insist that the indexed approach is the only reasonable way to define semantics for formal specifications. We have to be aware that the fibred approach can be more appropriate in some situations, especially, when it comes to software engineering.

Second, we have to understand the essential differences between the indexed and the fibred approach as well as the peculiarities of the fibred approach with regard to model-theoretic constructions and properties. We hope that our insight "fibred amalgamation = van Kampen" is helpful in this respect. As another small observation, not discussed in the paper, we want to mention that the translation of sentences along signature morphisms in an institution is often based on free constructions thus translation of sentences is more complex in the indexed setting while it is simple in the fibred setting.

We are convinced that we will need, at the end, a flexible and neat combination of indexed and fibred concepts and techniques to develop appropriate specification frameworks for MDSE. In other cases, like Hoare logic, for example, the best way to deal with semantics may be to use both fibred and indexed structures, in a way that the structural features of a framework are presented in an indexed manner and the features of deduction in a fibred one [24].

By the way, during our journey we have been missing sadly a comprehensive compendium on slice and comma categories.

Third, it may be reasonable, at some points, to weaken or to vary traditional categorical concepts and constructions to reach out for new applications, in a similar way as we weakened the concept "functor category" to the concept "interpretation category" and varied the Grothendieck construction.

We want to close the paper with a short reflection about the peculiarities of the fibred approach. Indexed amalgamation is easy since it relies on very strong assumptions about the identity of mathematical entities. All the categories Set, Rel, Graph, Cat, ... and thus also $[\mathsf{I} \to \mathsf{Cat}]$, Set/I, ... are, for example, build upon "extensional equality". The Grothendieck construction, however, tears us out from this ideal world by producing "copies" of mathematical entities, and there are no mechanisms or tools available within the fibred setting enabling us to control that two "copies" of the "same entity" behave in the "same way". They just become independent entities of their own.

Acknowledgments. The author wishes to thank the organizers of the special session "Mathematical Structures in Formal System Development and Analysis " of the Ninth Congress of Romanian Mathematicians in 2019 for inviting him to give a talk at the congress and to write this paper.

REFERENCES

[1] M. Barr and C. Wells, *Category Theory for Computing Science.* Series in Computer Science, London, Prentice Hall International, 1990.

[2] R. Diaconescu, *Institution-Independent Model Theory.* 1st ed., Birkhäuser Basel, 2008.

[3] Z. Diskin and U. Wolter, *A Diagrammatic Logic for Object-Oriented Visual Modeling.* Electronic Notes in Theoretical Computer Science **203** (2008), *6*, 19–41.

[4] Z. Diskin, *Databases as diagram algebras: Specifying queries and views via the graph-based logic of sketches.* Tech. Rep. **9602** , Frame Inform Systems, Riga, Latvia, http://www.cs.toronto.edu/ zdiskin/Pubs/TR-9602.pdf (1996).

[5] Z. Diskin, *Towards algebraic graph-based model theory for computer science.* Bulletin of Symbolic Logic **3** (1997), 144–145.

[6] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer, *Fundamentals of Algebraic Graph Transformations.* Berlin, Springer, 2006.

[7] H. Ehrig, M. Große-Rhode, and U. Wolter, *Applications of Category Theory to the Area of Algebraic Specification in Computer Science.* Applied Categorical Structures **6** (1998), *1*, 1–35.

[8] J. A. Goguen and R. M. Burstall, *Institutions: Abstract Model Theory for Specification and Programming.* Journals of the ACM **39** (1992), *1*, 95–146.

[9] R. Goldblatt, *Topoi: The Categorial Analysis of Logic.* Dover Publications, 1984.

[10] B. Jacobs, *Categorical Logic and Type Theory.* Elsevier, 1999.

[11] H. König and U. Wolter, *Van Kampen Colimits and Path Uniqueness.* Logical Methods in Computer Science **14** (2018), *2*, Article 5, 1–27.

[12] S. Lack and P. Sobociński, *Adhesive Categories.* In: I. Walukiewicz I (Ed.), *Proceedings of FOSSACS 2004*, pp. 273–288. Springer, LNCS 2987, 2004.

[13] F. Macías, A. Rutle, V. Stolz, R. Rodriguez-Echeverria, and U. Wolter, *An Approach to Flexible Multilevel Modelling.* Enterprise Modelling and Information Systems Architectures **13** (2018), Article 10, 1–35.

[14] F. Macías, U. Wolter, A. Rutle, F. Durán, and R. Rodriguez-Echeverria, *Multilevel coupled model transformations for precise and reusable definition of model behaviour.* Journal of Logical and Algebraic Methods in Programming **106** (2019), 167–195.

[15] M. Makkai, *Generalized Sketches as a Framework for Completeness Theorems.* Journal of Pure and Applied Algebra **115** (1997), 49–79, 179–212, 214–274.

[16] F. Mantz, G. Taentzer, Y. Lamo, and U. Wolter, *Co-evolving meta-models and their instance models: A formal approach based on graph transformation.* Science of Computer Programming **104** (2015), 2–43.

[17] A. Martini, U. Wolter, and E. H. Haeusler, *Fibred and Indexed Categories for Abstract Model Theory.* Logic Journal of the IGPL **15** (2007), *5-6*, 707–739.

[18] C. McLarty, *Elementary Categories, Elementary Toposes.* Oxford Logic Guides (Book 21). Clarendon Press, 1991.

[19] H. Reichel, *Initial Computability, Algebraic Specifications, and Partial Algebras.* Oxford University Press, 1987.

[20] A. Rossini, A. Rutle, Y. Lamo, and U. Wolter, *A formalisation of the copy-modify-merge approach to version control in MDE.* Journal of Logic and Algebraic Programming **79** (2010), *7*, 636–658.

[21] A. Rossini, J. de Lara, E. Guerra, A. Rutle, and U. Wolter, *A formalisation of deep metamodelling.* Formal Aspects of Computing **26** (2014), 1115–1152.

[22] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter, *A formal approach to the specification and transformation of constraints in MDE.* Journal of Logic and Algebraic Programming **81** (2012), *4*, 422–457.

[23] U. Wolter and Z. Diskin, *From Indexed to Fibred Semantics – The Generalized Sketch File.* Tech. rep. **361** (2007), Department of Informatics, University of Bergen.

[24] U. Wolter, A. R. Martini, and E. H. Haeusler, *Indexed and Fibred Structures for Hoare Logic.* Electronic Notes in Theoretical Computer Science **348** (2020), 125–145.

[25] U. Wolter and H. König, *Fibred Amalgamation, Descent Data, and Van Kampen Squares in Topoi.* Applied Categorical Structures **23** (2013), 447–486.

[26] U. Wolter, *An Algebraic Approach to Deduction in Equational Partial Horn Theories.* Journal of information processing and cybernetics. EIK **27** (1990), *2*, 85–128.

[27] U. Wolter, *Category Theory and Diagrammatic Modelling.* Script for the course INF223, Department of Informatics, University of Bergen, Norway, Draft (2020).

University of Bergen
Department of Informatics
P.O.Box 7803, N-5020 Bergen, Norway
`uwe.wolter@uib.no`