A categorical framework for the transformation of object-oriented systems: Models and data

Christoph Schulz, Michael Löwe, Harald König

Fachhochschule für die Wirtschaft Hannover, Freundallee 15, 30173 Hannover, Germany

Abstract

Refactoring of information systems is hard, for two reasons. On the one hand, large databases exist which have to be adjusted. On the other hand, many programs access those data. Data and programs all have to be migrated in a consistent manner such that their semantics does not change. This paper addresses the data part of the problem and introduces a model for object-oriented structures, describing the schema level with classes, associations, and inheritance as well as the instance level with objects and links. Positive Horn formulas based on predicates are used to formulate constraints to be obeyed by the schema and instance level, in order to reflect object-oriented structures. Homomorphisms are used for the typing of the instance level as well as for the description of refactorings which specify the addition, folding, and unfolding of schema elements. A categorial framework is presented which allows us to derive instance migrations from schema transformations in such a way that instances of the old schema are automatically migrated into instances of the new schema. The natural use of the pullback functor for unfolding is followed by an initial semantics approach: Instance migration is completed with the help of a co-adjoint functor on arrow categories.

1. Introduction

During the engineering and use of information systems, data and software undergo many modifications. These modifications can be divided into two categories. The first category contains all
modifications that have a direct and externally visible impact on the functionality of the software or on the information content of the database. The second category consists of modifications which only prepare modifications of the first category and which, by themselves, do not lead to changes in the behaviour of the software or in the meaning of the data under transformation. Modifications of the second category are called “refactorings” (Fowler, 1999). They provide a major method for quickly adapting software to constantly changing requirements.

Refactorings are expected to be applied multiple times in different but similar situations. This is comparable to the case of design patterns in software engineering which have emerged in the last twenty years (Gamma et al., 1995; Fowler, 2002). Consequently, a suitably general specification of a refactoring is necessary. This, however, requires a certain level of abstraction for the software and the data to be transformed. Such an abstraction is often called a schema or model and describes important structural aspects of the data and software, which are instances of, or typed in, this schema. Today, the “object-oriented view of life” dominates the field of software engineering. Therefore, models are typically object-oriented and try to capture the structure by grouping similar objects into classes and describing relations between them through associations of various types.

Two typical object-oriented refactorings are “Introduce a new superclass” (Fig. 1) and “Move the origin of an association from a subclass to a superclass”, as shown in Fig. 2. A combined application of these two refactorings on the schema in Fig. 3(a) could be used to prepare the model for an extension by an additional subclass of Customer, e.g. CorporateCustomer (Figs. 3(b) and (c)).

It is important to consider the consequences of a refactoring. Obviously, the more general the structures to be transformed, the greater the number of instances that are likely to be affected. Changing a data schema may not only require the data typed in this schema to be adjusted, but may also affect the software which uses the schema structures to access and manipulate the data. Changing a data schema typically requires one to adjust the whole system, i.e., existing data, the programs that handle the data, and the running processes (programs under execution). In this paper, we concentrate on the (induced) adjustment of the data. The mechanisms for programs and processes will be similar or straightforward extensions and are subjects for further research (compare Section 6).

We introduce a graph-like mathematical model which allows us to specify object-oriented data as well as schema refactorings. To describe data together with their schema, graph structures conforming to the model are used. Nodes of such graph structures represent classes (schema) or objects (instance); edges represent associations (schema) or links (instance). Homomorphisms between such graph

---

2 All class diagrams are specified in the UML (Fowler and Scott, 2003).
structures express typings, (parts of) refactorings, and migrations. Results of category theory are used to compute induced migrations from schema refactorings.

The paper is organized as follows. Section 2 incrementally introduces a graph structure specification MP with positive Horn formulas which constitutes the foundation of the mathematical description of object-oriented models and data. The category Alg(MP) of all MP-systems and MP-homomorphisms, as well as the (sub-)categories Alg(MP)↓S and Sys(S) with a fixed schema S, represent the universe of discourse for the following sections. Section 3 addresses the migration of data. The main result Theorem 11 shows that a functorial semantics can be provided that respects the properties of the object-oriented model. Section 4 provides a small case study with demonstrates the applicability of the theoretical model to practical situations. Section 5 compares the approach presented with other models for object-oriented systems. Section 6 gives an outlook on topics for future research.

2. Models and instances

The schema and the instance level of object-oriented systems are modelled by systems w.r.t. an extended specification.\(^3\) An extended specification Spec = (Σ, H) is an extended signature together with a set of positive Horn formulas H over a set of variables X. An extended signature Σ = (S, OP, P) consists of a set of sorts S, a family of operation symbols OP = (OP₁,...,OPₙ) \(\subseteq \Sigma \setminus S\), and a family of predicates P = (P₁,...,Pₙ) \(\subseteq \Sigma \setminus S\), such that \(s_i \in P_{s_i}\) for each sort \(s_i \in S\). A system A w.r.t. an extended signature Σ = (S, OP, P), for short a \(\Sigma\)-system, consists of a family of carrier sets (Aₖ), a family of operations \((\text{op}^A) : A_1 \to A_2\) \(\subseteq \Sigma \setminus S\), and a family of relations \((\text{p}^A) : A_1 \to A_2\) \(\subseteq \Sigma \setminus S\), such that \(\text{p}^A \subseteq A_i \times A_i\) is the diagonal relation for each sort \(s_i\). A system A w.r.t. an extended specification Spec = (Σ, H) is a \(\Sigma\)-system such that all axioms are valid in A. A \(\Sigma\)-homomorphism \(h : A \to B\) is a \(\Sigma\)-system \(h : A \to B\) between two \(\Sigma\)-systems A and B w.r.t. an extended signature \(\Sigma = (S, OP, P)\) is a family of mappings \(h_i : A_i \to B_i\), such that the mappings are compatible with the operations and relations, i.e., \(h_i \circ \text{op}^A \subseteq \text{op}^B \circ h_i\) for all operation symbols \(\text{op} : w \to s\) and \(h_i(w) \subseteq p^B\) for all predicates \(p : w\) where \(w = s₁s₂\ldots sₙ \in S^\ast\). Each \(\Sigma\)-homomorphism \(h : A \to B\) between two Spec-systems A and B w.r.t. an extended specification Spec = (Σ, H) is called a Spec-homomorphism.

The (first) model version for classes and instances is just graphs as depicted in Fig. 4. Nodes correspond to classes and edges correspond to associations. In Fig. 5(a), an exemplary UML schema is

---

\(^3\) See Maćzewski (1973) for the special case when signatures consist of one sort only.

\(^4\) Given \(w = s₁s₂\ldots sₙ\), \(A_i \subseteq \Sigma \setminus S\) is an abbreviation for the product set \(A₁ \times A₂ \times \cdots \times Aₙ\).

\(^5\) Given \(w = s₁s₂\ldots sₙ\), the term \(h_i(x₁, x₂, \ldots, xₙ)\) is shorthand notation for the term tuple \((h₁(x₁), h₂(x₂), \ldots, hₙ(xₙ))\).
Graph =
sorts
N (nodes)
E (edges)
opns
s : E \rightarrow N (source node of an edge)
t : E \rightarrow N (target node of an edge)

Fig. 4. The Graph signature.

(a) UML schema.
(b) Representing graph.
(c) Representing algebra.

Fig. 5. A system for the Graph signature.

MP_1 = Graph +
prds
under : N N (subnode of)
axms
inheritance
\[ x \in N : \text{under}(x, x) \] (reflexivity)
\[ x, y \in N : \text{under}(x, y) \land \text{under}(y, x) \Rightarrow x = y \] (antisymmetry)
\[ x, y, z \in N : \text{under}(x, y) \land \text{under}(y, z) \Rightarrow \text{under}(x, z) \] (transitivity)

Fig. 6. The MP_1 specification including the predicate under.

presented. The underlying graph for this schema is shown in Fig. 5(b). Fig. 5(c) illustrates the resulting Graph system.

An instance of a given schema S is represented as a system I w.r.t. the same signature, together with a typing homomorphism type : I \rightarrow S. At the instance level, nodes represent objects and edges constitute links.

The next model version provides the possibility of modelling inheritance relations between classes by an additional binary predicate under: If, in a system S, a class A is “under” a class B, i.e., if it is a subclass of B, then the relation under contains the pair (A, B). The specification MP_1 is shown in Fig. 6.\(^6\) As inheritance is hierarchical and, therefore, a partial order, it is reasonable to formulate corresponding requirements for the under relation.

While the use of the predicate under is quite natural at the schema level, the question arises of how it is to be interpreted at the instance level. Typically, objects are seen as monolithic entities even if they are mapped to multiple types in the class hierarchy. In this paper, we follow a different approach and consider objects to consist of a set of interconnected parts called particles. Each particle is represented by a node and is typed in a specific class in the schema. The advantage of this approach is that the structure of an object is made visible and resembles the object’s type hierarchy at the schema

\(^6\) The “MP” stands for “Model part” and describes the fact that systems for this model represent only a part (schema or instance) of the whole object-oriented system.
Fig. 7. Objects represented by particles.

Fig. 8. The \( MP_2 \) specification including the predicate \( rel \).

\[
MP_2 = MP_1 + \\
\text{prds} \\
rel : N \times N \\
\text{axms} \\
\text{components}
\]

\[
x, y \in N : rel(x, y) \Rightarrow rel(y, x) \quad \text{(symmetry)}
\]

\[
x, y, z \in N : rel(x, y) \land rel(y, z) \Rightarrow rel(x, z) \quad \text{(transitivity)}
\]

\[
x, y \in N : under(x, y) \Rightarrow rel(x, y) \quad \text{(components)}
\]

level allowing proper typing of links.\(^7\) Fig. 7 shows an exemplary instance level for the schema in Fig. 3(b).\(^8\)

The model currently allows an object to contain more than one particle for the same type. This is typically forbidden by object-oriented languages.\(^9\) In order to implement this requirement, we want to specify something like that:

\[
x, y \in N : rel(x, y) \land \text{type}(x) = \text{type}(y) \Rightarrow x = y \quad \text{(unique particles) (A1)}
\]

Here, another predicate \( rel \) has been used which will be fulfilled when two particles belong to the same object and, therefore, are related. Obviously, this predicate describes an equivalence relation as it is reflexive, symmetric, and transitive. Furthermore, the equivalence comprises the \( under \) relation, because each two particles connected by the \( under \) relation belong to the same object and, consequently, are part of the \( rel \) relation.\(^10\) The resulting specification \( MP_2 \) is shown in Fig. 8.\(^11\)

Another issue currently not resolved is association multiplicity: In our model, all associations are many-to-many, because the number of links at the instance level is not restricted in any way. However, many-to-one associations are often necessary in object-oriented schemas to allow at most one linked target object for any given association and source object. To achieve this, a formula like the following one is necessary, which disallows the existence of two links which are instances of the same association and start at the same particle\(^12\):

\[
x, y \in E : s(x) = s(y) \land \text{type}(x) = \text{type}(y) \Rightarrow x = y \quad \text{(at most one.) (A2)}
\]

---

\(^7\) For the purpose of typing, simple homomorphisms are sufficient; there is no need to introduce homomorphisms “up to inheritance”.

\(^8\) We do not use a different notation for schema inheritance and the relationship between particles because it can be easily deduced from the context which relation is meant.

\(^9\) An exception to this rule is the programming language \( C++ \) which explicitly allows this behaviour (International Organization for Standardization, 2003).

\(^10\) Note, however, that the \( rel \) relation might not be \text{generated} by the \( under \) relation. That means that there may exist related particles that do \text{not} belong to the same object. However, in our examples, this does not occur.

\(^11\) Note that reflexivity of the \( rel \) relation need not be specified by an axiom as it is a consequence from the combination of the first inheritance axiom and the last component axiom.

\(^12\) Note that this disallows multi-valued associations completely. This is desired, however, as only single-valued associations can be dereferenced at the instance level in a well-defined way. Multi-valued associations need further information (e.g. an index) when accessing links, which does not fit well in our graph structure model.
Note that even if multi-valued associations are prohibited by this axiom, it is nevertheless possible to link an object with a set of objects at the instance level by simulating many-to-many associations by many-to-one associations using recursive data structures (e.g. lists). This is shown in Fig. 9 where a data model using many-to-many associations is transformed into a structurally equivalent data model using many-to-one associations only.

The resulting model specification is summarized in the following definition:

**Definition 1 (Specification MP).** The specification $MP$ is defined as below:

$$MP = \begin{array}{l}
\text{sorts} \\
N \\
E \\
\text{opns} \\
s : E \rightarrow N \\
t : E \rightarrow N \\
\text{prds} \\
\under : N \times N \\
\rel : N \times N \\
\text{axms} \\
\begin{array}{ll}
x \in N : \under(x, x) & \text{(reflexivity)} \\
& \quad \quad \text{(A3)} \\
x, y \in N : \under(x, y) \land \under(y, x) \Rightarrow x = y & \text{(antisymmetry)} \\
& \quad \quad \text{(A4)} \\
x, y, z \in N : \under(x, y) \land \under(y, z) \Rightarrow \under(x, z) & \text{(transitivity)} \\
& \quad \quad \text{(A5)} \\
\text{components} \\
x, y \in N : \rel(x, y) \Rightarrow \rel(y, x) & \text{(symmetry)} \\
& \quad \quad \text{(A6)} \\
x, y, z \in N : \rel(x, y) \land \rel(y, z) \Rightarrow \rel(x, z) & \text{(transitivity)} \\
& \quad \quad \text{(A7)} \\
x, y \in N : \under(x, y) \Rightarrow \rel(x, y) & \text{(components)} \\
& \quad \quad \text{(A8)}
\end{array}
\end{array}$$

In order to make (A1) and (A2) proper axioms, we need to internalize the homomorphism type. To achieve this, we build a specification called $M$:

**Definition 2 (Specification M).** The specification $M$ consists of:

- two copies of $MP$, one for the schema part where the sorts, operation symbols, and predicates are suffixed by “S”, and one for the instance part, where the sorts, operation symbols, and predicates are suffixed by “I”;

![Fig. 9. Simulation of multi-valued associations.](image)
• two operation symbols

\[
\begin{align*}
\text{type } N & : NI \rightarrow NS \\
\text{type } E & : EI \rightarrow ES
\end{align*}
\]  

\text{(node typing)} 
\text{(edge typing)}

representing the typing;
• the axioms (A1) and (A2) with type replaced by type \( N \) in (A1) and by type \( E \) in (A2); and
• the homomorphism axioms below:

\begin{align*}
\text{typing is homomorphic} \\
x \in EI : type N(sl(x)) = sS(type E(x)) & \quad \text{(A9)} \\
x \in EI : type N(tl(x)) = tS(type E(x)) & \quad \text{(A10)} \\
x, y \in NI : under l(x, y) \Rightarrow under S(type N(x), type N(y)) & \quad \text{(A11)} \\
x, y \in NI : rel l(x, y) \Rightarrow rel S(type N(x), type N(y)) & \quad \text{(A12)}
\end{align*}

The axioms (A9)–(A12) are necessary to ensure that \((type N, type E)\) behaves like a homomorphism in every \( M \)-system. It should be evident that (A1) and (A2) are proper axioms in \( M \). In the sequel, these two axioms are called typing axioms.

We use the following notation: \( \text{Alg}(MP) \) denotes the category of all \( MP \)-systems and \( MP \)-homomorphisms; equivalently, \( \text{Alg}(M) \) denotes the category of all \( M \)-systems and \( M \)-homomorphisms. The objects of the arrow category \( \text{Alg}(MP)^2 \) are \( \text{Alg}(MP) \)-arrows \( I \xrightarrow{type} S \), which do not necessarily fulfill the typing requirements (A1) and (A2). The full subcategory \( \text{Sys} \subseteq \text{Alg}(MP)^2 \) restricts the arrow category to those arrows conforming to these requirements. Given a fixed schema system \( S \), the slice category \( \text{Alg}(MP) \downarrow S \) expresses the category of all \( \text{Alg}(MP) \)-arrows into the system \( S \), and the category \( \text{Sys}(S) \) denotes the full subcategory of \( \text{Alg}(MP) \downarrow S \) whose objects fulfill (A1) and (A2).\(^{13}\)

We are now able to express properly typed instances in two ways, either as an \( M \)-system or as an \( \text{Alg}(MP) \)-arrow in \( \text{Sys} \). Formally, these categories are isomorphic. In order to prove that, we first show that \( \text{Alg}(MP)^2 \) is isomorphic to \( \text{Alg}(M') \) where \( M' \) is defined as below.

**Definition 4** (Specification \( M' \)). The specification \( M' \) is \( M \) without the typing axioms (A1) and (A2). \( \square \)

**Lemma 4.** \( \text{Alg}(M') \) and \( \text{Alg}(MP)^2 \) are isomorphic.

**Proof.** We define the functors \( D : \text{Alg}(MP)^2 \rightarrow \text{Alg}(M') \) and \( P : \text{Alg}(M') \rightarrow \text{Alg}(MP)^2 \) which map \( \text{Alg}(MP)^2 \)-arrows to \( M' \)-systems and vice versa. For objects, we define \( D_{\text{Ob}}(type I : I \rightarrow S) ::= D \in \text{Ob}_{\text{Alg}(M')} \), where the carrier sets and operations of \( D \) are equal to the carrier sets and operations of \( I \) and \( S \) together with the internalized type operator, and \( P_{\text{Ob}}(D) ::= type I : I \rightarrow S \in \text{Ob}_{\text{Alg}(MP)^2} \), where the carrier sets and operations of \( D \) are split up into the two parts \( I \) and \( S \) and where \( type \) is externalized to \( type I \); note that the mapping of the typings is possible due to the homomorphism axioms. For morphisms, we have \( D_{\text{Mor}}((m, r) : (I \xrightarrow{type} S) \rightarrow (I' \xrightarrow{type'} S')) ::= m; r : D \rightarrow D' \) with \( D ::= D_{\text{Ob}}(type I), D' ::= D_{\text{Ob}}(type'I) \), and

\[
m; r(x) := \begin{cases} 
    m(x) & \text{if } x \in D_{NI} \cup D_{EI} \\
r(x) & \text{if } x \in D_{NS} \cup D_{ES}
\end{cases}
\]

\(^{13}\) Obviously, \( \text{Sys}(S) \) is also a subcategory of \( \text{Sys} \).
and \( \mathcal{P}_{\text{Mor}}(f : D \to D') \) \( \mathrel{::=} (m : I \to I', r : S \to S') : \text{type} \to \text{type}' \) with \( I \xrightarrow{\text{type}} S \mathrel{::=} \mathcal{P}_{\text{Ob}}(D), \) \( I' \xrightarrow{\text{type}'} S' \mathrel{::=} \mathcal{P}_{\text{Ob}}(D'), \) and

\[
\begin{align*}
m & \mathrel{::=} f \big|_{D_m \cup D_I} \\
r & \mathrel{::=} f \big|_{D_S \cup D_E}
\end{align*}
\]

It is easy to see that these functors are well-defined and that their composition yields the identity functor. \( \square \)

**Lemma 5.** \( \text{Alg}(M) \) and \( \text{Sys} \) are isomorphic.

**Proof.** This follows directly from Lemma 4. \( \square \)

For Horn clause specifications \( \text{Spec} = (\Sigma, H) \) where the signature only contains sorts and operation symbols, it is a well-known fact in universal algebra that the resulting category \( \text{Alg}(\text{Spec}) \) is closed under the formation of products and extremal subobjects (see e.g. Wechsler (1992, Theorem 14) for the single-sorted case). This result has been extended to signatures including predicates in Löwe et al. (2002). We prove in the Appendix that \( \text{Alg}(M) \) is a full and epireflective subcategory of \( \text{Alg}(M') \). This implies the following proposition:

**Proposition 6 (Epireflector \( \mathcal{F} \)).** There exists an epireflector \( \mathcal{F} : \text{Alg}(\text{MP})^2 \to \text{Sys} \).

**Proof.** By Proposition 31 (see the Appendix), \( \text{Alg}(M) \) is a full and epireflective subcategory of \( \text{Alg}(M') \). By Lemmas 4 and 5, \( \text{Sys} \) is a full and epireflective subcategory of \( \text{Alg}(\text{MP})^2 \). \( \square \)

Summarizing our results so far, an object-oriented schema is modelled as an \( \text{MP} \)-system \( S \). An instance of this schema consists of an \( \text{MP} \)-system \( I \) and a typing \( \text{MP} \)-homomorphism \( \text{type} : I \to S \) such that \( I \xrightarrow{\text{type}} S \) is an object of the category \( \text{Sys}(S) \). Every schema instance \( \text{type} : I \to S \) in \( \text{Alg}(\text{MP}) \downarrow S \) can freely be transformed into an object of the category \( \text{Sys} \) by the epireflector \( \mathcal{F} \).

### 3. Model transformation and data migration

So far we can describe object-oriented data. In this section we introduce schema transformations that can be uniquely extended to migrations of corresponding data.\(^{14}\)

**Definition 7 (Transformation, Refactoring).** A transformation \( t : S \xrightarrow{\text{\#}} S' \) in the category \( \text{Alg}(\text{MP}) \) is a span \( S \leftarrow S' \xleftarrow{\text{\#}} S' \). Such a transformation is called a refactoring iff \( \text{\#} \) is surjective. \( \square \)

A general transformation allows reduction and unfolding as well as extension and folding through the use of non-surjective homomorphisms (reduction and extension) and non-injective homomorphisms (unfolding and folding) on the left and right side of the span, respectively. Refactorings are special transformations which are constrained to surjective homomorphisms on the left side of the span. This constraint stems from the fact that refactorings are not allowed to delete schema objects because such a deletion almost always causes loss of information at the instance level, which contradicts the intuitive requirement that a refactoring preserve information. In the following we use the term schema transformation if the span consists of schema objects, and migration if the span consists of typed instances.

Given a typed instance \( I \xrightarrow{\text{type}} S \) and a schema transformation \( t : S \xrightarrow{\text{\#}} S' \), the migration is performed as follows (visualized in Fig. 10):

1. \( \mathcal{P}_{\text{\#}} \), the pullback functor along \( \text{\#} \), is applied to \( I \xrightarrow{\text{type}} S \), resulting in the typed instance \( I \xrightarrow{\text{type}\#} S' \).

This part of the transformation is responsible for unfolding instance elements if \( \text{\#} \) is not injective, and for deleting elements if \( \text{\#} \) is not surjective.

\(^{14}\) See Löwe et al. (2007, 2006) and König et al. (2007) for precursor material on data migration induced by schema transformations.
Let $I \xrightarrow{l^t} S_\#$ and $S_\# \xrightarrow{r^t} S'$. We know that $I \xrightarrow{l^t} S_\#$, resulting in the typed instance $I \xrightarrow{l^t} S_\#$. This part of the transformation is used to retype instance elements and to add new types without any instances.

(3) $I \xrightarrow{l^t} S'$ may violate the typing axioms. To fix this, we apply the epireflector $F : \text{Alg}(MP)^2 \to \text{Sys}$ to it, obtaining the typed instance $I' \xrightarrow{l^t} S'$ in the subcategory $\text{Sys}(S'')$ for a not yet known schema $S''$. This part of the transformation is responsible for identifying instance elements due to retyping.

The composition of the three functors results in the migration functor defined below:

**Definition 8 (Migration Functor).** Let $t : S \xrightarrow{\_} S'$ be a transformation. The migration functor $\mathcal{M}^t : \text{Sys}(S) \to \text{Sys}$ is defined as

$$\mathcal{M}^t := F \circ F'^t \circ F^t,$$

where the functor $F^t : \text{Sys}(S) \to \text{Alg}(MP) \downarrow S_\#$ is the pullback functor along $l^t$, the functor $F'^t : \text{Alg}(MP) \downarrow S_\# \to \text{Alg}(MP) \downarrow S'$ is the composition functor along $r^t$, and the functor $F : \text{Alg}(MP)^2 \to \text{Sys}$ is the epireflector into the subcategory $\text{Sys}$.

It is important to take a deeper look at the functors used to perform the migration. As noted above, the composition functor $F'^t$ need not preserve the typing axioms, such that the resulting system $I \xrightarrow{l^t} S'$ has to be adjusted by the epireflector $F$. The most crucial property of this epireflector is that it does not change the schema. This is important as it allows us to get rid of the unknown schema $S''$ introduced by the migration. The equality $S'' = S'$ is proven by the following lemma:

**Lemma 9 (Epireflector $F$ Does Not Change Schema).** Let $I \xrightarrow{l^t} S \in \text{Ob}\text{Alg}(MP)^2$ be given, and let $F_{\text{Ob}}(I \xrightarrow{l^t} S)$ be the typed instance $I' \xrightarrow{l^t} S'$. Then $S = S'$ holds.

**Proof.** We know that $(l' \xrightarrow{l^t} S', (u^S, u^I) : (I \xrightarrow{l^t} S) \to (l' \xrightarrow{l^t} S'))$ is free over $I \xrightarrow{l^t} S$, with $(u^S, u^I)$ being an $\text{Alg}(MP)$-epimorphism. That means that for each $\text{Sys}$-object $I' \xrightarrow{l^t} S'$ and an $\text{Alg}(MP)^2$-morphism $(p^S, p^I) : (I \xrightarrow{l^t} S) \to (I' \xrightarrow{l^t} S')$, there is a unique morphism $(v^S, v^I) : (l' \xrightarrow{l^t} S') \to (I' \xrightarrow{l^t} S')$ in $\text{Sys}$, such that $(v^S, v^I) \circ (u^S, u^I) = (p^S, p^I)$.

This fact is applied to $I' \xrightarrow{l^t} S' = S \xrightarrow{id_S} S$ and $(p^S, p^I) = (\text{type}_I, id_S)$, yielding the situation depicted in Fig. 11. Note that $S \xrightarrow{id_S} S$ trivially fulfills the typing axioms. From this we obtain

$$v^S \circ u^S = id_S \tag{1}$$

Furthermore, we have

$$u^S \circ v^S \circ u^S = u^S \circ id_S \tag{by (1)}$$

$$= u^S$$

$$= id_{S'} \circ u^S \tag{2}$$
We now show that \(u^S\) is an epimorphism in \(\text{Alg}(\text{MP})\). Let an \(\text{MP}\)-system \(X\) and two \(\text{MP}\)-homomorphisms \(p, q: S' \to X\) be given (Fig. 12) with
\[
p \circ u^S = q \circ u^S \tag{3}
\]
Then \((p, p \circ \text{type}_{I'})\) and \((q, q \circ \text{type}_{I'})\) are morphisms in \(\text{Alg}(\text{MP})^2\). Now we have
\[
p \circ \text{type}_{I'} \circ u^I = p \circ u^S \circ \text{type}_{I} \tag{left square commutes}
= q \circ u^S \circ \text{type}_{I} \tag{by (3)}
= q \circ \text{type}_{I'} \circ u^I \tag{left square commutes} \tag{4}
\]
From (3) and (4), we obtain
\[
(p, p \circ \text{type}_{I'}) \circ (u^S, u^I) = (q, q \circ \text{type}_{I'}) \circ (u^S, u^I) \tag{5}
\]
in the arrow category \(\text{Alg}(\text{MP})^2\). Because \((u^S, u^I)\) is epic in \(\text{Alg}(\text{MP})^2\), it follows from (5) that \((p, p \circ \text{type}_{I'}) = (q, q \circ \text{type}_{I'})\) and finally \(p = q\).

Because \(u^S\) is epic, we obtain from (2)
\[
u^S \circ v^S = \text{id}_{S'} \tag{6}
\]
From (1) and (6), we conclude that \(u^S\) is an isomorphism. Hence we can assume \(S = S'\) without loss of generality. \(\square\)

From this lemma, it follows that the functor \(\mathcal{F}\) can be restricted to a slice category for some fixed schema \(S\), resulting in a family of epireflectors \(\mathcal{F}^S: \text{Alg}(\text{MP}) \downarrow S \to \text{Sys}(S)\) for each possible schema \(S\). As noted before, this fact is important for the migration of data along a schema transformation, because this ensures that the migrated instances always conform to the target schema of the schema transformation.

Now, we show that in contrast to the composition functor \(\mathcal{F} \circ \mathcal{F}\), the pullback functor \(\mathcal{P}^\text{type}_{I'}\) does indeed preserve the typing axioms:

**Lemma 10 (Pullback Functor Preserves Typing Axioms).** Let \(f_S: S' \to S\) be an \(\text{MP}\)-homomorphism. Then the restriction of the pullback functor \(\mathcal{P}^\text{type}_{I'}: \text{Alg}(\text{MP}) \downarrow S \to \text{Alg}(\text{MP}) \downarrow S'\) to the domain \(\text{Sys}(S)\) has codomain \(\text{Sys}(S')\).

**Proof.** Let \(l \xrightarrow{\text{type}_{I'}} S \in \text{Ob}\text{Sys}(S)\) be a system conforming to the typing axioms. Applying the pullback functor \(\mathcal{P}^\text{type}_{I'}\) to it results in the system \(l' \xrightarrow{\text{type}_{I'}} S'\) of the category \(\text{Alg}(\text{MP}) \downarrow S'\) (Fig. 13).
According to Adámek et al. (2004), the system \( I' \) is an MP-system, as implicational categories are closed under the formation of products and extremal subobjects and, equivalently, under the formation of limits. It remains to show that the typing axioms are valid:

1. **Axiom (A1).** Let \( x, y \in I'_N \) be given with

   \[
   (x, y) \in \text{rel}^{I'}
   \]

   \[
   \text{type}_{I',N}(x) = \text{type}_{I',N}(y)
   \]

   We have to show that \( x = y \).

   As \( f^I \) is a MP-homomorphism,

   \[
   (f^I_N(x), f^I_N(y)) \in \text{rel}^{I'}
   \]

   follows from (7), and

   \[
   \text{type}_{I,N}(f^I_N(x)) = f^S_N(\text{type}_{I',N}(x)) = f^S_N(\text{type}_{I',N}(y)) = \text{type}_{I,N}(f^I_N(y))
   \]

   holds due to (8) and because the pullback diagram commutes. As \( I \xrightarrow{\text{type}_I} S \) fulfils the typing axioms, (9) and (10) imply

   \[
   f^I_N(x) = f^I_N(y)
   \]

   From (8) and (11), we obtain \( x = y \) due to the fact that pullback morphisms are jointly injective.

2. **Axiom (A2).** Let \( x, y \in I'_E \) be given with

   \[
   s'^I(x) = s'^I(y)
   \]

   \[
   \text{type}_{I,E}(x) = \text{type}_{I,E}(y)
   \]

   We have to show that \( x = y \).

   As \( f^I \) is a MP-homomorphism,

   \[
   s'(f^I_N(x)) = s'(f^I_N(y))
   \]

   follows from (12). The remainder of the proof is similar to (1).

Now we can formulate the main result of our paper:

**Theorem 11 (Migration Respects Typing Axioms).** Let \( I \xrightarrow{\text{type}_I} S \) be an object of the category \( \text{Sys}(S) \), and let a schema transformation \( t : S \xrightarrow{s^\#} S' \) be given. Then \( \mathcal{M}(I) \xrightarrow{\text{type}_I} S \in \text{Ob} \mathcal{S}'(S) \), i.e., the result of the migration is typed in the transformation’s target schema \( S' \) and respects the typing axioms (A1) and (A2).

**Proof.** According to Definition 7, the transformation \( t : S \xrightarrow{s^\#} S' \) corresponds to the span \( I \xrightarrow{\text{type}} S \xrightarrow{f^I} S' \). The first step of the migration according to Definition 8 is applying the pullback functor \( \mathcal{P}^I \) (cf. Fig. 10).

Pulling back the system \( I \xrightarrow{\text{type}_I} S \) along \( f^I \) results in the system \( I' \xrightarrow{\text{type}_I'} S' \). Due to Lemma 10, this system fulfils all \( M \)-axioms. The second step of the migration, the application of the composition functor \( \mathcal{F}^I \), yields the system \( I' \xrightarrow{\text{type}_I'} S' \). As this system does not necessarily fulfil the \( M \)-axioms, this system is adjusted via the epireflector \( \mathcal{F} \) in the last step, resulting in the system \( I'' \xrightarrow{\text{type}_I''} S'' \). Due to Lemma 9, we have \( S'' = S' \), such that the result of the migration is the system \( I'' \xrightarrow{\text{type}_I''} S' \). This system fulfils all \( M \)-axioms due to Proposition 6.  

![Fig. 13. Pullback situation.](image-url)
Corollary 12 (Schema Refactorings Induce Instance Refactorings). Let $I \xrightarrow{\text{type}} S$ be an object of the category $\text{Sys}(S)$, and let a schema refactoring $t : S \xrightarrow{\#} S'$ be given, i.e. the left span morphism $l^t$ is surjective. Then the induced instance transformation $t' : I \xrightarrow{l'} I'$ is a refactoring, as well.

Proof. We have to show that $l'$, the left-hand side of the induced instance transformation $t'$, is surjective. However, this follows from the fact that the pullback construction on the left side of the migration (see Fig. 10) preserves surjective homomorphisms.\footnote{See Goldblatt (1984) for details.}

4. Case study

Initial situation. A German insurance company offers third-party insurances for private and corporate customers. Customer data are stored within an object-oriented database. Each customer object consists of the customer's name, her address, and the associated insurance. Fig. 14 shows the schema as well as exemplary instances.
The notation used is as follows: At the schema level, rectangles denote classes (elements of $N$) and arrows stand for associations (elements of $E$). Attributes within classes are shorthand notation for associations to primitive classes. Inheritance between classes expressed by the under relation is shown using hollow-tip arrows; however, the reflexive and transitive pairs of the relation are not shown for clarity. The rel relation is not shown explicitly at all as it is expected to be generated by the under relation. At the instance level, rectangles represent object particles (elements of $N$) and hollow-tip arrows visualize the under relation. Consequently, each object in the sense of the paradigm of object-oriented programming is represented by a maximal set of particles connected directly or indirectly by hollow-tip arrows; this is underlined by using the same identifier for all particles belonging to the same object. Normal arrows denote links (elements of $E$). Values within particles are shorthand notation for links to instances of primitive classes. Like for the schema level, neither the rel relation nor the reflexive and transitive pairs of the under relation are visualized at the instance level.

Stage 1: Adding a new insurance product. Now, the insurance company plans to extend their portfolio by defence insurances. Consequently, the schema has to be adjusted accordingly. First, a new class Insurance is introduced as a superclass of ThirdPartyInsurance. Second, the attribute insuranceNumber is moved to this new class. Third, the target of the association product is changed to the new class Insurance. Finally, a new class DefenceInsurance is added as a subclass of Insurance. The resulting schema is shown in Fig. 15.

How can this schema transformation be described within our model? The introduction of the new superclass Insurance corresponds to an unfolding of a class by a non-injective morphism on the left side of the schema transformation, and is a special case of the more general schema transformation “Introduction of a superclass” (Fig. 16). The modification of objects and links by the induced migration is performed analogously. Note that the unfolding of instance particles on the left is due to the pullback construction.

Moving up the attribute insuranceNumber in the inheritance hierarchy must be modelled by a combination of unfolding on the left side and folding on the right side in order to ensure that the transformation mappings are homomorphisms (Fig. 17). On the left side, the class $B$ is unfolded, yielding the two classes $B$ and $X$ in the middle, and the origin of the association is moved to the temporary class $X$. On the right side the class $X$ is folded with the class $A$, such that the association starts at the class $A$ after the transformation. Again, the modification of objects and links by the induced migration is done similarly. Note that the folding of instance particles on the right side is due to the epireflector which takes care that axiom (A1) is satisfied.

Finally, the new class DefenceInsurance is simply added on the right side of a schema transformation, such that the left morphism is the identity and the right morphism an inclusion. Here, neither the pullback construction nor the epireflector has any impact on the instance level.

Stage 2: Supporting customers abroad. Until now, all customers of the insurance company have been from Germany. Now a customer from abroad arises. However, the schema is not yet prepared for this, because only German addresses can be stored. In order to support addresses abroad, the class Address is renamed to InlandAddress. Then, like in the last stage, the new superclass Address and the new class AbroadAddress which accepts a freeform address are created. Finally, the association address, which belongs to the class Customer, is retargeted at the new class Address. The resulting schema is displayed in Fig. 18. In our model, the necessary schema transformations and their effects on the instance level are similar to the ones described in the last stage, with the only exception that the end of an association is moved upwards instead of an attribute.

Stage 3: Concentration on corporate customers. Finally, the management decided to concentrate on corporate customers in the future and to abandon the business with private customers. After the contracts with private customers have been terminated, the schema has to accommodate for the change. First, the class PrivateCustomer is deleted. Second, the classes Customer and CorporateCustomer

\[\text{“Maximal” in the sense that no particle outside the set is connected to a particle within the set via a hollow-tip arrow.}\]

\[\text{Components unfolded by the left side of a schema transformation are framed with a dashed border, and components folded by the right side of a schema transformation are framed with a continuous border.}\]

\[\text{That is, Insurance and Address objects linked to PrivateCustomer objects have been removed.}\]
are merged to a single class CorporateCustomer. The resulting schema as well as the migrated instances are shown in Fig. 19. A change of the instance level is necessary in order to remove instances of deleted schema elements (here: objects of the class PrivateCustomer) and in order to fold particles.

On the schema level, two different schema transformations have to take place in order to achieve the described effect. The deletion of the class PrivateCustomer is performed by a schema transformation where the left morphism is a proper inclusion which does not reach the class node; the right side is simply the identity. By the pullback construction, objects typed in the class PrivateCustomer are deleted as well. The merge of the two classes CorporateCustomer and Customer is a special case of the general schema transformation “Merging two related classes” shown in Fig. 20, where classes are folded on the right side of the schema transformation by a non-injective morphism on the right side. Note that simple deletion of both PrivateCustomer and CorporateCustomer classes and renaming of Customer is not sufficient, as the class CorporateCustomer may contain further associations and attributes (not shown here) such that the deletion would lead to a data loss at
the instance level. Alternatively, if CorporateCustomer is neither deleted nor merged with Customer, unused “zombie” particles remain at the instance level, which is clearly undesirable, too.

Note that this migration causes any remaining PrivateCustomer object to be retyped to become a CorporateCustomer object. If this is not desired, PrivateCustomer objects have to be deleted manually before applying the schema transformation and instance migration.

5. Related work

There exist algebraic models for object-oriented data and program structures, e.g. Ehrig et al. (2005, 2006) and Kastenberg et al. (2006a,b). However, to our knowledge, our model is different in that it represents object structures at the instance level as conglomerates of separate particles. Although this approach, called object slicing, has already been described in Kuno et al. (1995) and Young-Gook and Rundensteiner (1997), it has not been used within an algebraic or categorial framework yet. The great advantage of object slicing is that the instance level can be typed in the schema by using standard homomorphisms. Other models (e.g. the models based on “node type inheritance” Ehrig et al. (2005, 2006)) need to develop special typing constructs (“attributed clan morphisms” in this case) in order to respect the semantics of inheritance. We chose the object slicing approach as in our opinion, the complexity of object-oriented models is more easily dealt with when put into the object structure rather than into the (typing) homomorphism. Additionally, the object slicing view represents a common technique in object-oriented software development when class inheritance is implemented by delegation. Finally, our approach can be extended by an operational model; see Schulz et al. (2009, 2010) and Schulz (2010) for details.

Last but not least, our approach is unique in the respect that it combines a model for data with a model for schema transformations and induced migrations. To our knowledge, no algebraic models of object-oriented data exist which specifically support schema transformation as well as induced data migration.
6. Outlook

The contributions of this paper can be summarized as follows: The framework presented proposes part of an approach towards a more human-independent migration of object-oriented systems. The innovative part of the theory presented is the use of algebraically specified arrow categories that foster simultaneous transformation of model and data with the help of a functor proven to act on suitable comma categories. The functor codes the automatic computation of a migration target from a given typed instance based on schema transformation rules.

This functor is composed of three components: The pullback functor $P^{\#}$ causes unfolding and adding of instances, the retyping functor $F^{-r}$ is known to be its co-adjoint (Goldblatt, 1984), and,
finally, the co-adjoint functor $F^S$ into the subcategory $\text{Sys}(S)$ is responsible for identifying objects that will equally be typed after the migration.

Thus, the whole migration enjoys well-understood universal properties which can further be pursued: State changes of object-oriented programs can be formalized by co-limits in graph-based categories—as e.g. in the DPO approach (Corradini et al., 2004; Kastenberg et al., 2006a,b). Hence, it is an important question under which circumstances the whole migration functor has a right-adjoint such that it preserves co-limits. If so, transformation rules induce simultaneous adaptation of methods and running programs. Obviously, a sufficient condition for the migration to be an adjunction is that $\mathcal{P}^d$ enjoys this property. McLarty (1995) proves the existence of a right-adjoint of $\mathcal{P}^d$ in cartesian closed categories, but more careful investigations might yield more cases of adjointness of the whole migration.

A second direction for future research is the development of tools that support migration induced by refactoring rules. If transformation rules can be captured ergonomically in an appropriate application, migrations can automatically and uniquely (by adjointness) be computed. Thus, content migration of databases, programs and processes is possible. These tools should discover the potential for composition, as well: Bigger refactorings should be decomposable into elementary changes;
atomic steps must be proved to be composable to more comprehensive procedures. Determining a complete set of simple atomic refactorings is another facet for future research.

Furthermore, correctness in Theorem 11 is based on initial semantics. This approach must be compared with formal specifications of “information” to distinguish between semantics-preserving refactorings and information-distorting transformations (Miller et al., 1994). In the same way, behavioural semantics of programs have to be used to underpin transformation procedures of methods and processes.

Appendix

In this section we prove that, given two extended specifications Spec = (Σ, Φ) and Spec′ = (Σ, Φ′) with Φ ⊆ Φ′ over some extended signature Σ = (S, OP, P), Alg(Spec) is an epireflective subcategory of Alg(Σ) (Proposition 30) and Alg(Spec′) is an epireflective subcategory of Alg(Spec) (Proposition 31).

Definition 13 (Full Homomorphisms). Let Σ = (S, OP, P) be an extended signature, let h: A → B be a Σ-homomorphism, and let p ∈ Pw be a predicate over w ∈ S*. Then h is full on p if

\[ h_w(x) \in p^B \Rightarrow x \in p^A \]  \hspace{1cm} (15)

for all x ∈ Aw, h is full if h is full on every predicate in Σ.

Definition 14 (Jointly Full Homomorphisms). Let Σ = (S, OP, P) be an extended signature, let \((f^i): A → B^i\)i∈I be a family of Σ-homomorphisms, and let p ∈ Pw be a predicate for some w ∈ S*. Then \((f^i)\)i∈I are jointly full on p if

\[ (\forall i \in I : f^i_w(x) \in p^{B^i} \Rightarrow x \in p^A) \]  \hspace{1cm} (16)

for all x ∈ Aw, \((f^i)\)i∈I are jointly full if \((f^i)\)i∈I are jointly full on every predicate in Σ.

Lemma 15 (Full Homomorphisms). Let Σ = (S, OP, P) be an extended signature and p ∈ Pw be a predicate for some w ∈ S*.

1. If \((f^i): A → B^i\)i∈I is a jointly full family of Σ-homomorphisms on p and g: X → A is another Σ-homomorphism full on p, then the family of Σ-homomorphisms \((f^i \circ g): X → B^i\)i∈I is also jointly full on p.
2. If f: A → B and g: B → C are two Σ-homomorphisms, and g \circ f: A → C is full on p, then f is full on p.
3. The identity id^A: A → A is full on p for each Σ-system A.

Definition 16 (Forgetful Functor V). Let Σ = (S, OP, P) be an extended signature. Then let V: Alg(S, OP, P) → Alg(S, OP) be the forgetful functor with V_{ob}(A) = A' and V_{mor}(f) = f, where A' is identical to A on the carrier sets and operations.

In the following, if Σ = (S, OP, P) is an extended signature, we denote by Σ' its restriction to sets and operation symbols, i.e. Σ' = (S, OP).

Lemma 17 (Forgetful Functor V is Adjoint). The forgetful functor V: Alg(Σ) → Alg(Σ') according to Definition 16 has a left-adjoint.

Proof. Any system A ∈ Alg(Σ') can be extended by an empty family of relations, yielding the system \(\mathcal{F}(A) \in \text{Ob} Alg(\Sigma')\). Obviously, \((\mathcal{F}(A), id^A)\) is an Alg(Σ)-reflection for A. Combining this with \(\mathcal{F}(f) = f\), we have a free functor \(\mathcal{F}: Alg(\Sigma') → Alg(\Sigma)\). □

Lemma 18 (Monomorphisms in Alg(Σ)). Let Σ = (S, OP, P) be an extended signature. A Σ-homomorphism h: A → B is monic iff it is injective.

Proof. For the “if”-part, note that because h is already a Σ'-monomorphism and each Σ-homomorphism is also a Σ'-homomorphism, the test for being monic holds in Alg(Σ) as well. For the “only-if”-part, recall that adjoint functors preserve monomorphisms (Adámek et al., 2004, Proposition 18.9). So, given a Σ-monomorphism f, V(f) is a monomorphism due to Lemma 17. As monomorphisms are injective in Alg(Σ'), f is injective. □

Lemma 19 (Epimorphisms in Alg(Σ)). Let Σ = (S, OP, P) be an extended signature. A Σ-homomorphism h: A → B is epic iff it is surjective.
Proof. For the “if”-part, note that because $h$ is already a $\Sigma'$-epimorphism and each $\Sigma'$-homomorphism is also a $\Sigma'$-homomorphism, the test for being epic holds in $\text{Alg}(\Sigma)$ as well. We prove the “only-if”-part by contraposition. Thus, let $h$ be non-surjective. As $V$ is the identity on morphisms, $V(h)$ is not surjective either. Because epimorphisms and surjective homomorphisms coincide in $\text{Alg}(\Sigma')$, it follows that $V(f)$ is not epic. Therefore, there are two $\Sigma'$-homomorphisms $g, h: V(B) \to C'$ for some $\Sigma'$-system $C'$ with $g \neq h$ and $g \circ V(f) = h \circ V(f)$. Now we construct the $\Sigma$-system $C$ by extending $C'$ by the missing relations such that $C$ makes all predicates true. This allows us to reinterpret $g$ and $h$ as $\Sigma$-homomorphisms. As $V(f) = f$, we finally have $g \neq h$ and $g \circ f = h \circ f$ in $\text{Alg}(\Sigma)$. So $f$ is not epic. \hfill $\square$

Corollary 20 (Isomorphisms in $\text{Alg}(\Sigma)$). Let $\Sigma = (S, OP, P)$ be an extended signature. A $\Sigma$-homomorphism $h: A \to B$ is an isomorphism if and only if it is full.

Proof. The “only-if”-part follows directly from Lemma 15, (2) and (3). It remains to prove the “if”-part. The corresponding properties of $\text{Alg}(\Sigma')$-morphisms guarantee the existence of a $\Sigma'$-isomorphism $h^{-1}: B \to A$ with $h^{-1} \circ h = id_B$ and $h \circ h^{-1} = id_A$. It remains to show that $h^{-1}$ preserves relations. Let $p \in P_w$ be some predicate with $w \in S^*$, and let some $x \in p^B$. By assumption, $h$ is surjective. So there exists a $y \in A_w$ with $h_w(y) = x \in p^B$. By assumption, $h$ is full, which implies $y \in p^A$. Thus we have $h^{-1}_w(x) = h^{-1}_w(h_w(y)) = y \in p^B$. \hfill $\square$

Lemma 21 (Extremal Monomorphisms in $\text{Alg}(\Sigma)$). Let $\Sigma = (S, OP, P)$ be an extended signature. A $\Sigma$-monomorphism $h: A \to B$ is extremal if and only if it is full.\footnote{A monomorphism $m: A \to B$ is called extremal if for each factorization $m = f \circ e$ with $e$ being an epimorphism, $e$ is an isomorphism.}

Proof. First, we prove the “if”-part. Let $h$ be a full monomorphism. Let $h = m \circ e$ be some factorization with the $\Sigma$-epimorphism $e$: $A \to C$ and the $\Sigma$-morphism $m$: $C \to B$. According to Lemma 19, $e$ is surjective. According to Adámek et al. (2004, Proposition 7.34 (2)), $e$ is a $\Sigma$-monomorphism and thus injective (Lemma 18). Finally, $e$ is full due to Lemma 15(2). By Corollary 20, $e$ is an isomorphism.

We prove the “only-if”-part by contraposition. Thus, let $h$ be a $\Sigma$-monomorphism that is not full. So there is a predicate $p \in P_w$ with $w \in S^*$ and some $x \in A_w$ with $x \notin p^A$ and $h_w(x) \in p^B$. As $\text{Alg}(\Sigma')$ has a unique (Epi, Epi)-factorizations, we have $V(h) = m \circ e$ with $\Sigma$-epimorphism $e$: $V(A) \to C'$ and a $\Sigma'$-monomorphism $m$: $C' \to V(B)$ for some $\Sigma'$-system $C'$. We now construct the $\Sigma$-system $C$ as an extension of $C'$ such that for every predicate $p' \in P_w'$ with $w' \in S^*$, we have $y \in p'^C \Leftrightarrow m_w'(y) \in p'^B$ for all $y \in C_w'$. As $h$ preserves relations, this definition ensures that $e$ and $m$ also preserve relations. Therefore, $e$ and $m$ are $\Sigma$-homomorphisms. However, $e$ is not full: By definition of $C$, $e_w(x) \in p^B$, but $x \notin p^A$. So $e$ is not a $\Sigma$-isomorphism. \hfill $\square$

Lemma 22 (Unique (Epi, ExtrMono)-Factorization in $\text{Alg}(\Sigma)$). Let some extended signature $\Sigma = (S, OP, P)$ be given. Then $\text{Alg}(\Sigma)$ has unique (Epi, ExtrMono)-factorizations, i.e. each $\Sigma$-homomorphism $h: A \to B$ can be split into a $\Sigma$-epimorphism $e: A \to C$ and an extremal $\Sigma$-monomorphism $m: C \to B$ for some $\Sigma$-system $C$ such that $h = m \circ e$, and for every other (Epi, ExtrMono)-factorization $h = m' \circ e'$ with $e': A \to C'$ and $m': C' \to B$, there exists a $\Sigma$-isomorphism $u: C \to C'$ with $e' = u \circ e$ and $m = m' \circ u$.

Proof. (Existence) As $\text{Alg}(\Sigma')$ has unique (Epi, Mono)-factorizations, we have $V(h) = m \circ e$ with $\Sigma$-epimorphism $e$: $V(A) \to C'$ and a $\Sigma'$-monomorphism $m$: $C' \to V(B)$ for some $\Sigma'$-system $C'$. We now construct the $\Sigma$-system $C$ as an extension of $C'$ such that for every predicate $p \in P_w$ with $w \in S^*$, we have $x \in p^C \Leftrightarrow m_w(x) \in p^B$ for all $x \in C_w$. As $h$ preserves relations, this definition ensures that $e$ and $m$ also preserve relations. Therefore, $e$ and $m$ are $\Sigma$-homomorphisms and, by Lemmas 18 and 19, $m$ is monic and $e$ is epic in $\text{Alg}(\Sigma)$. By definition, $m$ is full. By Lemma 21, it follows that $A \xrightarrow{e} C \xrightarrow{m} B$ is an (Epi, ExtrMono)-factorization of $h$ in $\text{Alg}(\Sigma)$.

(Uniqueness) Let $A \xrightarrow{e} C \xrightarrow{m} B$ and $A \xrightarrow{e'} C' \xrightarrow{m'} B$ be two (Epi, ExtrMono)-factorizations of $h$. Because $\text{Alg}(\Sigma')$ has unique (Epi, Mono)-factorizations, we conclude the existence of a $\Sigma'$-isomorphism...
Let \( C \rightarrow C' \) with \( u \circ e = e' \) and \( m' \circ u = m \). It is sufficient to show that \( u \) preserves relations since Lemma 15(2) guarantees that it is full. Let \( p \in P_w \) be some predicate with \( w \in S^* \), and let some \( x \in C_w \) be given. Due to the surjectivity of \( e \), there is a \( y \in A_w \) with \( e_w(y) = x \). We conclude that \( x \in p^C \Rightarrow e_w(y) \in p^C \Rightarrow m_w(e_w(y)) \in p^B \Rightarrow m'_w(u_w(e_w(y))) \in p^B \Rightarrow u_w(e_w(y)) \in p^{C'} \Rightarrow u_w(x) \in p^{C'} \).

**Construction 23** (Products in \( \text{Alg}(\Sigma) \)). Let \( \Sigma = (S, OP, P) \) be an extended signature. Let \( (A^i)_{i \in I} \) be an \( I \)-indexed family of \( \Sigma \)-systems. Then we construct the product system \( \times A \) and the family \( (\pi^i : \times A \rightarrow A^i)_{i \in I} \) of projections as in \( \text{Alg}(\Sigma') \), extended by the relations for all predicates \( p \in P_w \) with \( w = s_1 s_2 \ldots s_n \in S^* \) and \( n \in \mathbb{N}_0 \) as follows:

\[
p^{\times A} ::= \{(x^1_{i \in I}, x^2_{i \in I}, \ldots, x^n_{i \in I}) \mid (x^1_j, x^2_j, \ldots, x^n_j) \in p^A \text{ for all } j \in I \}
\]

The projections preserve relations: For each predicate \( p \in P_w \) with \( w = s_1 s_2 \ldots s_n \in S^* \), \( n \in \mathbb{N}_0 \) and for all \( x = ((x^1_{i \in I}, x^2_{i \in I}, \ldots, x^n_{i \in I}) \in A_w \), we have

\[
((x^1_{i \in I}, x^2_{i \in I}, \ldots, x^n_{i \in I}) \in p^{\times A} \Rightarrow (x^1_j, x^2_j, \ldots, x^n_j) \in p^A \\
\Rightarrow \pi_w((x^1_{i \in I}, x^2_{i \in I}, \ldots, x^n_{i \in I}) \in p^A) \]

**Lemma 24** (Products in \( \text{Alg}(\Sigma) \)). \( \times A, (\pi^i : \times A \rightarrow A^i)_{i \in I} \) from Construction 23 is the categorical product of an \( I \)-indexed family of \( \Sigma \)-systems \( (A^i)_{i \in I} \) in \( \text{Alg}(\Sigma) \).

**Proof.** Let \( B \) be a \( \Sigma \)-system and \( (p^i : B \rightarrow A^i)_{i \in I} \) be a family of \( \Sigma \)-homomorphisms. As \( \text{Alg}(\Sigma') \) has products, there exists a \( \Sigma' \)-homomorphism \( u : \mathcal{V}(B) \rightarrow \mathcal{V}(\times A) \) with \( \mathcal{V}(p^i) = \mathcal{V}(\pi^i) \circ u \) for all \( i \in I \). By definition, \( \mathcal{V}(f) = f \) for all \( \Sigma \)-homomorphisms \( f \), so we only need to show that \( u \) preserves relations, making it a \( \Sigma \)-homomorphism. For each predicate \( p \in P_w \) with \( w = s_1 s_2 \ldots s_n \in S^* \), \( n \in \mathbb{N}_0 \) and for all \( x = (x_1, x_2, \ldots, x_n) \in B_w \), we have

\[
x \in p^B \Rightarrow p^i_w(x) \in p^A_i \text{ for all } i \in I \\
\Rightarrow ((p^i_{x_1}(x_1)), (p^i_{x_2}(x_2)) \ldots, (p^i_{x_n}(x_n))_{i \in I}) \in p^{\times A} \\
\Rightarrow u_w(x) \in p^{\times A}
\]

**Lemma 25** (Projections are Jointly Full). Let \( \times A, (\pi^i)_{i \in I} \) be product of \( (A^i)_{i \in I} \) in \( \text{Alg}(\Sigma) \). Then the projections \( (\pi^i)_{i \in I} \) are jointly full.

**Proof.** A direct consequence of the definitions of the relations in \( \times A \).  

**Lemma 26** (Homomorphisms Preserve Solutions). Let \( \Sigma = (S, OP, P) \) be an extended signature, and let \( \phi = (X : p(t)) \) be an atomic \( \Sigma \)-formula over a variable set \( X \) with \( t \in T^X(X)_w \) and \( p \in P_w, w \in S^* \). Let \( f : A \rightarrow B \) be a \( \Sigma \)-homomorphism and \( \text{asg} : X \rightarrow A \) be a variable assignment in \( A \). Then \( f \circ \text{asg} \) solves \( \phi \) in \( B \) whenever \( \text{asg} \) solves \( \phi \) in \( A \).

**Proof.** By Ehrig and Mahr (1985, Theorem 3.3(1)), we have \( f \circ \text{eval}(\text{asg}) = \text{eval}(f \circ \text{asg}) \). It follows that \( \text{eval}(\text{asg})_w(t) \in p^A \Rightarrow f_w(\text{eval}(\text{asg})_w(t)) \in p^B \Rightarrow \text{eval}(f \circ \text{asg})_w(t) \in p^B \).

**Lemma 27** (Full Homomorphisms Reflect Solutions). Let an extended signature \( \Sigma = (S, OP, P) \) be given, and let \( \varphi = (X : p(t)) \) be an atomic \( \Sigma \)-formula over a variable set \( X \) with \( t \in T^X(X)_w \) and \( p \in P_w, w \in S^* \). Let \( f : A \rightarrow B \) be a full \( \Sigma \)-homomorphism and \( \text{asg} : X \rightarrow A \) be a variable assignment in \( A \). Then \( \text{asg} \) solves \( \varphi \) in \( A \) whenever \( f \circ \text{asg} \) solves \( \varphi \) in \( B \).

**Proof.** By Ehrig and Mahr (1985, Theorem 3.3(1)), we have \( f \circ \text{eval}(\text{asg}) = \text{eval}(f \circ \text{asg}) \). It follows that \( \text{eval}(f \circ \text{asg})_w(t) \in p^B \Rightarrow f_w(\text{eval}(\text{asg})_w(t)) \in p^B \).
Lemma 28 (Closure Under Formation of Full Subsystems). Let Spec = (Σ, Φ) be an extended specification over some extended signature Σ = (S, OP, P). Then Alg(Spec) is closed under the formation of full subsystems in Alg(Σ).

Proof. Let A be a Spec-model and (E, e: E → A) be a full Σ-subsystem of A. Let asg: X → E be a variable assignment in E that solves all premises of some axiom φ = (X : (prei)i∈I) ⇒ con) ∈ Φ in X. We have to show that asg also solves the conclusion con in E. By Lemma 26, the variable assignment e ◦ asg solves the premises of φ in A. Because A is a Spec-model, e ◦ asg solves the conclusion of φ in A. Since e is full, the variable assignment asg solves the conclusion of φ in E by Lemma 27.

Lemma 29 (Closure Under Formation of Products). Let Spec = (Σ, Φ) be an extended specification over an extended signature Σ = (S, OP, P). Then Alg(Spec) is closed under the formation of products in Alg(Σ).

Proof. Let asg: X → ×A be a variable assignment in ×A which solves all premises of some axiom φ = (X : (prei)i∈I) ⇒ con) ∈ Φ in X. We have to show that asg also solves the conclusion con in ×A. Obviously, pr i ◦ asg is a variable assignment in A i for all i ∈ I. As asg solves all premises of φ in ×A, we conclude by Lemma 26 that pr i ◦ asg solves the premises of φ in A i for all i ∈ I. By assumption, (A i)i∈I are Spec-models. Thus, pr i ◦ asg solves the conclusion of φ in A i for all i ∈ I. Since, by Lemma 25, the projections (pr i)i∈I are jointly full, asg solves the conclusion of φ in ×A.

Proposition 30 (Full and Epireflective Subcategories (1)). Let Spec = (Σ, Φ) be an extended specification over some extended signature Σ = (S, OP, P). Then Alg(Spec) is a full and epireflective subcategory of Alg(Σ).

Proof. By definition, Alg(Spec) is full subcategory of Alg(Σ). By Lemmas 28 and 29, Alg(Spec) is closed under the formation of full subsystems and products in Alg(Σ). By Lemma 21, Alg(Spec) is closed under the formation of extremal subobjects. By Lemma 22, Alg(Σ) has unique (Epi, ExtrMono)-factorizations. By Adámek et al. (2004, Theorem 16.8), we conclude that Alg(Spec) is an epireflective subcategory of Alg(Σ).

Proposition 31 (Full and Epireflective Subcategories (2)). Let any two extended specifications Spec = (Σ, Φ) and Spec′ = (Σ, Φ′) with Φ ⊆ Φ′ be given over some extended signature Σ = (S, OP, P). Then Alg(Spec′) is a full and epireflective subcategory of Alg(Spec).

Proof. Alg(Spec′) and Alg(Spec) are full subcategories of Alg(Σ) by definition. Furthermore, all Spec-axioms are valid in each Spec′-model by assumption; thus Alg(Spec′) is a full subcategory of Alg(Spec).

Now let A be some Spec-model. By Proposition 30, Alg(Spec′) is a full and epireflective subcategory of Alg(Σ). Thus there is an epireflector F : Alg(Σ) → Alg(Spec′) and a Alg(Spec′)-epireflection (FA, uA) for A in Alg(Σ). We have to show that (FA, uA) is also a Alg(Spec′)-epireflection for A in Alg(Spec). Let B be some Spec′-model and f: A → B a Spec-homomorphism. As Alg(Spec′) is an epireflective subcategory Alg(Σ) by Proposition 30, there exists a unique Spec′-homomorphism f′: FA → B such that f′ = f ◦ uA holds. As all these Σ-homomorphisms are also Spec-homomorphisms, the equation also holds in Alg(Spec). Thus (FA, uA) is also a Alg(Spec′)-reflection for A in Alg(Spec).

References


---

20 A subsystem A ▸ B is full if m is full.