On the Role of Category Theory in the Area of Algebraic Specifications

H. Ehrig, M. Große–Rhode, and U. Wolter

Technische Universität Berlin, FB Informatik, Sekr. FR 6–1
Franklinstr. 28/29, D – 10587 Berlin, Germany

e–mail: {ehrig,mgr,wolter}@cs.tu–berlin.de

Abstract. The paper summarizes the main concepts and paradigms of category theory and explores some of their applications to the area of algebraic specifications.
In detail we discuss different approaches to an abstract theory of specification logics. Further we present a uniform framework for developing particular specification logics. We make use of 'classifying categories', to present categories of algebras as functor categories and to obtain necessary basic results for particular specification logics in a uniform manner. The specification logics considered are: equational logic for total algebras, conditional equational logic for partial algebras, and rewrite logic for concurrent systems.

1 Category Theory and Applications in Computer Science

Category theory has been developed as a mathematical theory over 50 years and has influenced not only almost all branches of structural mathematics but also the development of several areas of computer science. It is the aim of this paper to review the ideas and concepts of category theory and to discuss their role in theoretical computer science.

Especially for readers not familiar with, but interested in, category theory this section contains an introductory survey of the main concepts and paradigms of category theory. Further we present three selected problems in the area of algebraic specifications. In the subsequent sections we show how these problems can be solved using concepts, constructions and results of category theory. These sections provide a concise exposition of the application of more advanced category theory in the area of (algebraic) specifications thus they become harder to read for novice readers. But nevertheless interested readers can use these sections as an appropriate guideline for approaching category theory. Abstracting from the examples we summarize in the conclusion the role of category theory in theoretical computer science, and formulate some general concepts.

1.1 Paradigm and Concepts of Category Theory

Category theory arose from the fundamental idea of representing a function by an arrow. This idea first appeared in topology about 1940 (see [Mac71]). The original purpose of category theory in the pioneering paper [EM45] by EILENBERG and MACLANE was to establish a uniform framework to speak about isomorphisms and natural equivalences appearing in different areas of mathematics, in particular algebra and topology.

What was new about category theory? Within set theory we can characterize a mathematical object only by describing its inner structure, that is, we can separate it into different parts and elements and we can represent the interrelations between these inner components by means of ordered tuples.

Category theory takes the opposite viewpoint. Relevant are the *morphisms*, that is, the connections between the object in question and the other objects around. This means, we try to characterize a certain object by its specific role within the net of relationships among all objects under consideration.

The crucial discovery of category theory, that many kinds of constructions within different areas of mathematics, like conjunction/intersection/product, disjunction/union/direct sum, kernel, quotient, implication/function space, can be described in a uniform way by those specific roles (*universal properties*) without reference to the inner structure, had a great impact on mathematics, and, particularly, on theoretical computer science.

The abstract viewpoint of category theory to look only at the *morphisms* between objects and not inside the objects has two obvious consequences. Firstly, we can describe objects only *up to isomorphism*, that is, independently from a particular representation or implementation. Often this is very nice and convenient, but especially in computer science the problem of an appropriate representation has to be solved in the end to make the theoretical results applicable. Secondly, category theory is mainly devoted to problems with a rich and complex structure of relationships between objects.

The informal idea of morphisms between objects leads naturally to the formal concept of a category: A **category** $\mathcal{C}$ consists of a class $Obj_{\mathcal{C}}$ of **objects** and a class $Mor_{\mathcal{C}}$ of **morphisms (arrows)**. Further a function $dom_{\mathcal{C}} : Mor_{\mathcal{C}} \to Obj_{\mathcal{C}}$ assigns to each morphisms f its **domain (or source)** $dom_{\mathcal{C}}(f)$ and a function $cod_{\mathcal{C}} : Mor_{\mathcal{C}} \to Obj_{\mathcal{C}}$ assigns to each morphisms f its **codomain (or target)** $cod_{\mathcal{C}}(f)$. Conveniently we write $f : A \to B$ for $f \in Mor_{\mathcal{C}}$ with $dom_{\mathcal{C}}(f) = A$ and $cod_{\mathcal{C}}(f) = B$. $Mor_{\mathcal{C}}(A, B)$ denotes the class of all morphisms $f : A \to B$. For each object $A \in Obj_{\mathcal{C}}$ there is assumed to be an **identity** morphism $id_A : A \to A$ and for any two morphisms $f : A \to B$ and $g : B \to C$, that is, morphisms f and g with $cod_{\mathcal{C}}(f) = dom_{\mathcal{C}}(g)$, there exists the **composition** $g \circ f : A \to C$. Composition is associative and the identities are just the identities with respect to composition, that is, $f \circ id_A = f$ and $id_B \circ f = f$ for all morphisms $f : A \to B$.

A popular example of a category is the category $\mathcal{SET}$ of sets, where $Obj_{\mathcal{SET}}$ is the class of all sets and $Mor_{\mathcal{SET}}$ is the class of all total functions between set. But considering partial functions or relations instead of total functions we obtain two further categories $\mathcal{PAR}$ and $\mathcal{REL}$, also with sets as objects.

Structured sets as objects — for several kinds of structures, including algebraic, topological, and order theoretic structures — and structure preserving functions as morphisms yield further important examples of categories. Here objects (structured sets) are organized *into categories* (via structure preserving functions) to allow their abstract characterization through universal properties.

But in many cases the structured sets itself, like ordered sets, monoids, algebraic theories or propositional logics (see [LS86]), can be considered *as categories*. A partially ordered set (or pre-ordered set) $(C, \leq)$, for instance, can be seen as a category $\mathcal{C}$ with $Obj_{\mathcal{C}} = C$ and $Mor_{\mathcal{C}} = \{(c,d) \mid c \leq d\}$, that is, $Mor_{\mathcal{C}}(c,d)$ is empty in case $c \not\leq d$ and $Mor_{\mathcal{C}}(c,d)$ is a singleton set in case $c \leq d$. A monoid $(M, \cdot, e)$ gives rise to a category $\mathcal{M}$ with one object m, identity $id_m = e$, and $Mor_{\mathcal{M}} = M$ where $\cdot$ is considered to be the composition in $\mathcal{M}$.

We have defined categories as *graphs* extended by identity and composition (see [Mac71, BW85]). But it is worth mentioning that the concept 'category' can also be seen as a refinement of the concept 'pre-ordered set'. Pre-ordered sets only represent the existence of a connection between two objects. But with categories we are able to distinguish between different possibilities of connecting two objects. Besides the interpretation of pre-ordered sets as categories we can abstract from any category $\mathcal{C}$ a pre-ordered set $(Obj_{\mathcal{C}}, \leq)$ with $A \leq B$ if there exists a morphism $f : A \to B$ in $\mathcal{C}$, that is, $Mor_{\mathcal{C}}(A, B) \neq \emptyset$.

Category theory allows to characterize mathematical constructions by universal properties. As an example we consider the *cartesian product* of two sets A and B. From the categorical viewpoint the construction provides not only the object $A \times B$ but also two *projections* $\pi_A : A \times B \to A$ and $\pi_B : A \times B \to B$. Moreover we have for free the *tupling* of functions: For any set C and any functions $f : C \to A$ and $g : C \to B$ there exists a unique function $h : C \to A \times B$ with $\pi_A \circ h = f$ and $\pi_B \circ h = g$. Because h is uniquely determined by f and g according to the two conditions we can denote h without ambiguity by $\langle f, g \rangle$. For the particular case $f = \pi_A$ and $g = \pi_B$ the uniqueness of h entails $\langle \pi_A, \pi_B \rangle = id_{A \times B}$. Following this observation we can prove that the *universal property* of tupling characterizes the cartesian product $A \times B$ in the category $\mathcal{SET}$ uniquely up to isomorphism.

A morphism $f : A \to B$ in a category $\mathcal{C}$ is an **isomorphism** if there exists a (unique) morphism $g : B \to A$ with $g \circ f = id_A$ and $f \circ g = id_B$. The isomorphisms in $\mathcal{SET}$ are precisely the bijections.

What are the benefits from the categorical characterization of cartesian product? Firstly, it turns out that many results in set theory can be shown on the abstract level of morphisms by using universal properties. That is, we get rid of tedious element chasing in proofs. Secondly, we can relate the concept 'cartesian product' to constructions in other categories, that is, to constructions with the same universal property. In partially ordered sets, for instance, the intersection (meet) of two elements has the same universal property, and for propositional logics the conjunction corresponds to the cartesian product. Thirdly, we can establish a strong relationship between the concepts 'cartesian product' and 'disjoint union'. The disjoint union $A + B$ of two sets A and B is characterized by a universal property that is *dual* to the universal property of the cartesian product: For any set C and any functions $f : A \to C$ and $g : B \to C$ there exists a unique function $h : A + B \to C$ with $h \circ i_A = f$ and $h \circ i_B = g$. Thereby

$i_A : A \to A+B$ and $i_B : B \to A+B$ are the embeddings of A and B respectively into $A+B$.

Categorical duality is the process 'Reverse all arrows' (see [Mac71]). The **dual** of any statement φ about categories is formed by making the following replacements throughout φ: 'domain' by 'codomain', 'codomain' by 'domain' and 'h is the composite of f with g' by 'h is the composite of g with f, that is, arrows and composites are reversed.

Above we have seen that product (cartesian product) and coproduct (disjoint union) are dual concepts. Other pairs of dual concepts are terminal/initial object, monomorphism/epimorphism, and many others.

An object T is **terminal (final)** in a category $\mathcal{C}$ if for each object A in $\mathcal{C}$ there is exactly one morphism $A \to T$. If T is terminal, the only morphism $T \to T$ is the identity thus any two terminal objects of $\mathcal{C}$ are isomorphic in $\mathcal{C}$. An object I is **initial** in a category $\mathcal{C}$ if for each object A in $\mathcal{C}$ there is exactly one morphism $I \to A$. Any two initial objects of $\mathcal{C}$ are isomorphic in $\mathcal{C}$. The empty set is initial in $\mathcal{SET}$, $\mathcal{PAR}$, and $\mathcal{REL}$ and is terminal in $\mathcal{PAR}$ and $\mathcal{REL}$. Any singleton set is terminal in $\mathcal{SET}$. For a many-sorted signature $\Sigma = (S, OP)$ the ground term algebra T_Σ is initial in the category $Mod(\Sigma)$ of all total Σ-algebras and all Σ-homomorphisms (see [EM85]). A terminal object in $Mod(\Sigma)$ is a Σ-algebra with a singleton set for each sort in S.

A morphism $m : A \to B$ is **monomorphic** in $\mathcal{C}$ when for any two parallel morphisms $f, g : C \to A$ the equality $m \circ f = m \circ g$ implies $f = g$; in other words m is monomorphic if it can always be cancelled on the left. The monomorphisms in $\mathcal{SET}$ are precisely the injections, and the epimorphims, that is, the right cancellable morphisms, are the surjections.

If we use morphisms as the basic concept to reason about problems it becomes quite natural to require, that a construction is not merely a function from objects of one species to objects of another species, but must also preserve the essential relationships among these objects [Gog91]. This intuition leads to the concept of a functor F between categories $\mathcal{C}$ and $\mathcal{D}$. A **functor** $F : \mathcal{C} \to \mathcal{D}$ is given by two suitably related mappings between the objects and morphisms of $\mathcal{C}$ and $\mathcal{D}$: The *object mapping* $F_{Obj} : Obj_{\mathcal{C}} \to Obj_{\mathcal{D}}$ and the *morphism mapping* $F_{Mor} : Mor_{\mathcal{C}} \to Mor_{\mathcal{D}}$ have to be compatible with the domain and codomain functions in $\mathcal{C}$ and $\mathcal{D}$, that is, $F_{Mor}(f) : F_{Obj}(A) \to F_{Obj}(B)$ for each morphism $f : A \to B$ in $\mathcal{C}$. Moreover, a functor F has to preserve composition and identities, that is, $F_{Mor}(g \circ f) = F_{Mor}(g) \circ F_{Mor}(f)$ and $F_{Mor}(id_A) = id_{F_{Obj}(A)}$.

Trivial examples of functors are the embeddings of $\mathcal{SET}$ into $\mathcal{PAR}$ and of $\mathcal{PAR}$ into $\mathcal{REL}$. In the area of algebraic specifications the so-called **forgetful functors**, that is, functors provided by *forgetting structure*, are important. For instance, we obtain a forgetful functor $|\_| : Mod(\Sigma) \to \mathcal{SET}$ by assigning to each Σ-algebra A the carrier $|A|$, that is, the underlying S-indexed family of sets. More generally any signature inclusion $\Sigma_1 \subseteq \Sigma_2$ gives rise to a forgetfull functor $U : Mod(\Sigma_2) \to Mod(\Sigma_1)$. Thereby we obtain for any Σ_2-algebra A the Σ_1-*reduct* $U(A)$ by just forgetting all components of A that disappear in Σ_1.

Other typical examples of functors are provided by constructions like cartesian products. For that reason we have to extend the cartesian product to functions: For any two functions $f : A \to C$ and $g : B \to D$ we obtain a function $f \times g : A \times B \to C \times D$ if we set $f \times g = \langle f \circ \pi_A, g \circ \pi_B \rangle$, that is, $(f \times g)(a, b) = (f(a), g(b))$ for all $(a, b) \in A \times B$. Now the property $(h \times k) \circ (f \times g) = (h \circ f) \times (k \circ g)$ ensures that the construction of cartesian products can be considered as a functor $\times : \mathcal{SET} \times \mathcal{SET} \to \mathcal{SET}$ from the product category $\mathcal{SET} \times \mathcal{SET}$ into the category $\mathcal{SET}$. For two categories $\mathcal{C}$ and $\mathcal{D}$ the objects in the **product category** $\mathcal{C} \times \mathcal{D}$ are pairs $(C, D) \in Obj_{\mathcal{C}} \times Obj_{\mathcal{D}}$ and the morphisms are pairs $(f, g) \in Mor_{\mathcal{C}} \times Mor_{\mathcal{D}}$. Identity and composition are defined componentwise, that is, $id_{(C,D)} = (id_C, id_D)$ and $(h, k) \circ (f, g) = (h \circ f, k \circ g)$.

One of the problems where the advantage of the categorical viewpoint becomes evident is the universal formalization of the intuitive concept of canonical construction. Given an object A in a category $\mathcal{C}$ we want to equip canonically this object with an additional structure, thus obtaining an object $F(A)$ in a category $\mathcal{D}$. Firstly, canonically means that this construction has to be compatible with morphisms, that is, it can be extended to a functor $F : \mathcal{C} \to \mathcal{D}$. Secondly, the construction should be uniform for all objects, that is, the net of relationships between the objects in $\mathcal{C}$ should be preserved by the construction, and there must not be additional relationships. To express this requirement we have to relate the situation in $\mathcal{D}$ with the former situation in $\mathcal{C}$. The crucial observation for solving this problem is, that in most cases of canonical constructions there is a simple functor $U : \mathcal{D} \to \mathcal{C}$ provided by dropping the additional structure of the objects and morphisms in $\mathcal{D}$. Now the strong preservation of relationships can be expressed by requiring a bijection between the sets $Mor_{\mathcal{C}}(A, U(B))$ and $Mor_{\mathcal{D}}(F(A), B)$ of morphisms for all objects A in $\mathcal{C}$ and B in $\mathcal{D}$. More precisely, we require the following **universal property** of the canonical construction: For each object A in $\mathcal{C}$ there is a unique morphism $\eta_A : A \to U(F(A))$, such that for each object B in $\mathcal{D}$ and each morphism $f : A \to U(B)$ in $\mathcal{C}$ there is a morphism $f^* : F(A) \to B$ in $\mathcal{D}$, uniquely determined by the property $U(f^*) \circ \eta_A = f$. It is important, that this requirement determines the canonical construction up to isomorphism. If we have such an **adjoint situation** F is called a **free** or **left adjoint** functor and U is also referred to as a **right adjoint** functor according to the position of F and U respectively in the bijection between the sets of morphisms above. The notion of adjoint functors is a key categorical concept and it turns out that adjoint situations are omnipresent in mathematics and theoretical computer science.

A familiar example for a free functor is the construction of term algebras $T_\Sigma(X)$ over sets X of variables. We have $\mathcal{C} = \mathcal{SET}$, $\mathcal{D} = Mod(\Sigma)$, and $U : Mod(\Sigma) \to \mathcal{SET}$ is the forgetfull functor $|\_|$ described above. $\eta_X : X \to |T_\Sigma(X)|$ is the inclusion of X into the set of terms over X, and the universal property means that any variable assignment $\alpha : X \to |A|$ can be extended uniquely to a homomorphism $\overline{\alpha} : T_\Sigma(X) \to A$ such that the restriction of $\overline{\alpha}$ to X equals α. That is, the universal property subsumes the principle of structural induction (algebraic recursion [Wec92]). More generally there exists for any inclusion $SPEC_1 \subseteq SPEC_2$ of (conditional) equational specifications a functor

$F : Mod(SPEC_1) \to Mod(SPEC_2)$ left adjoint to the restricted forgetfull functor $U : Mod(SPEC_2) \to Mod(SPEC_1)$. This result is essential in the area of algebraic specifications because the free functors F provide semantics of parameterized specifications.

To avoid the wrong impression that right adjoint functors are necessarly forgetful functors we refer to the product functor $\times : \mathcal{SET} \times \mathcal{SET} \to \mathcal{SET}$ that turns out to be right adjoint to the *diagonal functor* $\Delta : \mathcal{SET} \to \mathcal{SET} \times \mathcal{SET}$ with $\Delta_{Obj}(A) = (A, A)$ and $\Delta_{Mor}(f) = (f, f)$.[1] Note that the attributes 'left adjoint' and 'right adjoint' are relative, that is, one and same functor can be as well 'left adjoint' as 'right adjoint' but in different adjoint situations. For instance the diagonal functor turns out to be right adjoint with respect to the functor $+ : \mathcal{SET} \times \mathcal{SET} \to \mathcal{SET}$ provided by disjoint union.

Besides the characterization of single objects and the description of more or less concrete constructions on objects (and morphisms) we are faced in computer science also with the problem of comparing and relating different constructions. In particular, in many cases it will be necessary to speak about the equivalence of two concrete constructions, for instance, the equivalence between the polish postfix notation and the tree representation of terms. Category theory proposes the concept of natural transformation for this purpose. Given two parallel functors $F, G : \mathcal{C} \to \mathcal{D}$ a **natural transformation** $\eta : F \Rightarrow G$ is a mapping which assigns to each object A in $\mathcal{C}$ a morphism $\eta_A : F(A) \to G(A)$ in $\mathcal{D}$ relating the two results of applying F and G respectively to A. Again, these relations have to be compatible with morphisms, that is, we require $\eta_B \circ F(f) = G(f) \circ \eta_A$ for every morphisms $f : A \to B$ in $\mathcal{C}$. Note that a natural transformation η does not affect the whole target category $\mathcal{D}$. η only chooses some morphisms from $\mathcal{D}$ to relate suitably the images of $\mathcal{C}$ with respect to F and G respectively. Typical examples are provided by adjoint situations, where we have a natural transformation $\eta : Id_{\mathcal{C}} \Rightarrow U \circ F$ between the identity functor on $\mathcal{C}$ and the composition $U \circ F$ of functors. That is, $\eta_A : A \to U(F(A))$ describes where the original object A can be found in the free extension $F(A)$.

These basic concepts of category theory arise quite naturally from its paradigm which emphasises looking only at the morphisms between objects and not inside the objects.

1.2 General Remarks concerning the Role of Category Theory

Reflecting our own experiences made by approaching, using, and teaching category theory we want to identify four essential roles of category theory in computer science which are obviously mutually dependent. For a more comprehensive

[1] To show this assertion directly it is more convenient to use the following dual and equivalent definition of an adjoint situation: For each object B in $\mathcal{D}$ there is a unique morphism $\varepsilon_B : F(U(B)) \to B$, such that for each object A in $\mathcal{C}$ and each morphism $g : F(A) \to B$ in $\mathcal{D}$ there is a morphism $g^\bullet : A \to U(B)$ in $\mathcal{C}$, uniquely determined by the property $\varepsilon_B \circ F(g^\bullet) = g$.

informal discussion of the usefulness of category theory in computer science we refer to the very nice "Categorical Manifesto" of J.A. Goguen [Gog91].

Notions and Language Like set theory category theory also provides a universal conceptual language to speak formally and informally about the problems and situations we are faced with in computer science. The basic concepts have the same simplicity as the basic concepts of set theory, and they provide the necessary complementary viewpoint to deal properly with problems in computer science.

Besides the basic concepts we have also in mind concepts like initial object, terminal object, monomorphism, and epimorphism, which are the categorical counterparts of the set theoretical notions empty set, singleton set, injection, and surjection. But there are many other categorical concepts, like Hom-functors, adjoint functors, pullbacks and pushouts, limits and colimits, comma categories and functor categories (see [Mac71, BW85, BW90]) worth including as an integrated part of the common language of computer scientists.

Abstraction and Generalization The impulse for abstraction naturally appears if we get the feeling that we have essentially done the same work twice in defining concepts or proving results. Category theory offers an appropriate level of abstraction and a well established set of concepts for formulating the common essence of different things. For instance, the categorical product turns out to be the common essence of the concepts cartesian product of sets, intersection (meet) of partially ordered elements, and conjunction of propositions. In the same way the categorical coproduct covers the concepts disjoint union, union (join), and disjunction.

From a suitable categorical description of concepts and results we can potentially find advantages in two directions. Firstly, a categorical restatement of results and proofs can deliver a clear modularization of a particular theory. The idea is to divide the theory into an abstract categorical part and a concrete problem specific part with a well-defined interface between the two parts. The concrete part investigates the specific features of the concrete problem and the interface consists of a categorical description of these specific features. Based on the concepts and propositions in the interface the abstract part then develops the desired results by categorical arguments only. A modularization in this sense opens the opportunity for a *horizontal generalization*. That is, we can solve similar problems by retaining the general abstract part and by proving that the new problems also meet the requirements of the interface.

Secondly, we can look for a *vertical generalization* covering more concepts and providing deeper insights into the structure of our problem. Following this line products and terminal objects, for instance, generalize to limits on one side, and coproducts and initial objects generalize to colimits on the dual side. Limits then also cover the concepts preimage and equations, and colimits cover the concepts rewriting and quotient respectively.

Methodological Guideline Besides being a guideline for generalization category theory turns out to be also very helpful in the study of new areas in the stage before concepts and results have been developed. That means, in fields that are not yet well understood and developed category theory provides support for formulating appropriate basic concepts and for clarifying the structure of problems, that is, for asking the right questions.

Universal Constructions and Results This point finally addresses the potential technical benefit from category theory. That is, after establishing the connection between a concrete problem and the categorical world we get access to a wide spectrum of universal concepts, constructions, results, and theories probably helpful in solving our concrete problem. Important concepts and constructions are, for instance, filtered colimits, cartesian and monoidal closed categories, functor and comma categories, indexed and fibred categories, 2-categories, Kleisli categories, and Hom-functors. Useful results are the Yoneda lemma, results concerning the composition and decomposition of limits or colimits respectively, the preservation of limits by Hom-functors, and the preservation of limits (colimits) by right (left) adjoint functors. As further areas of category theory that have a direct influence on computer science we want to mention topos and sheaf theory [Joh77], algebraic theories [Law63, Man76], theory of sketches [BW85], and categorical logic [KR77, LS86].

Sections 3, 4, and 5 will elucidate how a horizontal generalization applying universal constructions and results from category theory works.

1.3 Interrelations between Category Theory and Algebraic Specifications

In this subsection we want to give a short overview how concepts and problems in the area of algebraic specification can be related to categorical concepts and constructions.

Models and Semantics Any formal specification will have both a syntactic part and a semantic part. In the area of algebraic specifications the syntactic part is given by a specification $SPEC$ of a variety of algebras (for example, a signature together with a set of equational axioms) with, perhaps, some additional information that distinguishes certain parts of the signature.

For algebraic specifications there are, at least, three ways to give a semantics to a specification, $SPEC$. Which one is appropriate depends on the purpose of the specification:

If we view the specification as a set of requirements that we want the intended models to satisfy but don't regard it as a complete description, then there may be many possible models satisfying these requirements. In this case the categorical paradigm suggests that the collection of models should form a subcategory of the category $Mod(SPEC)$ of all $SPEC$-algebras and the homomorphisms between them. The most common choices are the category $Mod(SPEC)$ itself, called

loose semantics, and the category $Mod_{gen}(SPEC)$ of all *generated* $SPEC$-algebras. A $SPEC$-algebra A is **generated** if A has no proper subalgebra. In terms of category theory generated can be expressed in two equivalent ways: The unique homomorphism $i_A : I \to A$ from the initial $SPEC$-algebra I to A is epimorphic, or A is preinitial (semi-initial), that is, for any other $SPEC$-algebra B there is at most one morphism $f : A \to B$. Note that the restriction to generated algebras becomes necessary if we want to use inductive proofs for reasoning about the intended models.

If we view the specification as a complete description of the properties of the intended models, then there is an intuitive sense in which we will get only one model. Clearly though, any specific model will have specific details which could be modified without really changing the essentials of the model, for example, the representation of natural numbers by binaries or decimals, or the representation of terms by polish postfix notation or by trees. This idea is captured categorically by saying that the models form an isomorphism class. Category theory also suggests some reasonable choices for the desired isomorphism class.

One is to take the class of all initial algebras in $Mod(SPEC)$. This can be viewed as a positive interpretation of the axioms, that is, saying that two data items are equal only if they are forced to be so by the axioms (**no confusion**) and that all data items are generated by the constants and operations (**no junk**) [MG85].

Another is to take the class of all terminal algebras in a suitable subcategory of $Mod(SPEC)$ defined by syntactic and semantic restrictions. In particular we have to take from $Mod(SPEC)$ the terminal algebras, since the terminal algebras in $Mod(SPEC)$ are trivial one-element algebras. This amounts to a specification of a machine rather than a data type [MG85] where we are looking for fully abstract models, that is, two internal states have to be identified as long as there is no observation making them distinguishable. In this sense the specification will be interpreted negatively.

Parameterization A generic data type provides a canonical way of extending pre-given data types by new sorts and/or new operations, that is, a generic data type turns out to be a "functor" producing for any suitable actual data type a unique extended data type.

To cover this idea in the area of algebraic specifications we can introduce the concept of parameterized specifications. A **parameterized specification** is given by a parameter specification PAR and a body specification BOD such that $PAR \subseteq BOD$. PAR is devoted to specify the 'suitable actual data types', that is, we want to represent the class of all suitable actual data types by the category $Mod(PAR)$. Further the part of BOD not contained in PAR is devoted to specify the new sorts and operations, that is, the extended data types are considered to be BOD-algebras. Recalling our discussion of functors in subsection 1.1 we can finally choose a free functor $F : Mod(PAR) \to Mod(BOD)$, if it exists, as the appropriate semantics of the parameterized specification.

In case a parameterized specification describes a generic data type that has been defined before by a set theoretical construction that gives rise to a functor $G : Mod(PAR) \to Mod(BOD)$, we could ask for the correctness of the parameterized specification with respect to the given generic data type. **Correctness** then means, that there exists a natural isomorphism $\eta : F \Rightarrow G$, that is, each component $\eta_A : F(A) \to G(A)$ of the natural transformation η is an isomorphism.

At least if we are thinking about actualization of parameterized specifications, where we have to cover also renaming and similar things, it becomes convenient to make the step from inclusions of specifications to **specification morphisms**. Then it appears that specifications and their morphisms constitute a category $\mathcal{SPEC}$, and that actualization syntactically means the construction of **pushouts** in this category. Following this line we can further consider the model categories $Mod(SPEC)$ as objects in a category $\mathcal{CAT}$ of all categories and functors. Since there is a forgetful functor $Mod(h) : Mod(SPEC_2) \to Mod(SPEC_1)$ for each specification morphism $h : SPEC_1 \to SPEC_2$ the definition of loose semantics yields a **model functor** $Mod : \mathcal{SPEC} \to \mathcal{CAT}^{op}$ from the category $\mathcal{SPEC}$ of specifications into the opposite of the category of all categories. The twist of the direction of morphisms is defined for any category $\mathcal{C}$ as its **opposite category** $\mathcal{C}^{op}$, given by $Obj_{\mathcal{C}^{op}} = Obj_{\mathcal{C}}$ and $Mor_{\mathcal{C}^{op}}(A, B) = Mor_{\mathcal{C}}(B, A)$ for all $A, B \in Obj_{\mathcal{C}^{op}}$.

For a proper concept of actualization we require that *Mod* maps pushouts in $\mathcal{SPEC}$ to **pullbacks** in $\mathcal{CAT}$. This result is known as amalgamation lemma (see [EM85]).

Structuring and Refinement Algebraic specifications are particularly useful to deal with problems in the large (see [EM90]), especially horizontal and vertical structuring.

Let us first consider concepts for horizontal structuring and modularization. These arise naturally from the categorical viewpoint. A structured specification is just given by a diagram of small specifications and specification morphisms between them, and the corresponding composed large specification will be the colimit of this diagram. Furthermore the syntactical construction should be compatible with the semantics, that is, the model functor has not only to map pushouts to pullbacks, but also arbitrary (finite) colimits to (finite) limits.

Secondly we have to deal with vertical structuring, that is, with refinement and implementation. Syntactically we have again diagrams of specifications. But semantically we have to take into account not only forgetful functors, but also free functors, and have to show compatibility with suitable limit and colimit constructions.

Logic of Specifications There are at least two reasons to look for an abstract theory of specification logics. Firstly, we want to generalize horizontally results obtained for one specification logic , like total algebras with equations, to another specification logic, like partial algebras with conditional existence equations. Secondly, it could be necessary to change the specification logic to obtain more specification power or to use theorem provers from other logics. In this situation we need a conceptual level, where we can speak about the relations between specification logics.

For this purpose the almost equivalent concepts of indexed and fibred categories are very useful. An **indexed category** is simply a functor $C : \mathcal{I} \to \mathcal{CAT}^{op}$ from a category $\mathcal{I}$ into the opposite of the category of all categories, that is, the objects in $\mathcal{I}$ are used to index the objects in $\mathcal{CAT}$. The model functor $Mod : \mathcal{SPEC} \to \mathcal{CAT}^{op}$ is a typical example. In the same way specifications or sets of formulas can be considered to be indexed by signatures.

Examples for abstract theories of specification logics are the theories of institutions [GB92], logical systems [Mes89], and specification frames [Ehr89, EG94].

Categorical methods are nowadays used in many different areas. A survey on further applications in the area of algebraic specifications can be found found in [EM95].

1.4 Selected Algebraic Specification Problems solved by Category Theory

In the following we present three representative algebraic specification problems which have been solved using category theory. These solutions will be discussed in more detail in the following sections. The first problem is closely related to the concept of logic of specifications discussed above.

Problem 1 (Independence of Specification Logic) The basic notion of an algebraic specification $SPEC = (S, OP, E)$ includes sort symbols, operation symbols, and equations as axioms and the models are total algebras (see [EM85]). Meanwhile several variations of syntax and semantics for algebraic specifications have been considered, like partial, order sorted, continuous, behavioural, and higher order specifications with different kinds of axioms.

The problem is to formulate the main concepts of syntax, semantics, and deduction, and some of their relevant properties for all these different algebraic specification formalisms, in a uniform way and independent of the specific specification logic.

Problem 2 (Theory of Partial Algebras) In most applications of algebraic specifications it turns out that a lot of the operations in the application domain are in fact partial. To allow this partiality and not to force explicit error handling in the beginning of the specification process, partial algebras instead of total ones should be employed as data type models. Then the problem arises to design a specification logic that extends equational specifications and total algebras to the partial case in a natural way, such that the following basic results are retained:

- Initial Objects,
- Sound and Complete Deduction Calculus,
- Free Functor Semantics,
- Compositionality.

Problem 3 (Specification Logic for Concurrency) The formal specification of concurrent, state dependent systems is an important task when open distributed processing is addressed, and basically, any software component that is placed into an environment is of this kind. Thus concepts that seem to be far beyond the scope of algebraic (data type) specification have to be taken into account. The problem is, similar to problem 2, to find a specification logic for concurrency that retains as much as possible of the theory of algebraic specification (see above).

The categorical solution of *problem 1* will be presented in the following section. In Section 3 we describe the concept of *classifying categories* which will be used in Sections 4 and 5 to give the solutions of *problem 2* and *3* respectively.

2 Abstract Theory of Specification Logics

In order to solve *problem 1* we must show how properties of specification logics can be described without reference to a particular notion of specification or model. The most important results of algebraic specifications for instance, that will also be addressed in the subsequent sections, can be stated in this 'specification logic independent' style in natural language as follows.

- The existence of *prototypes*: For each specification there is a model that embodies exactly the information of the specification, characterized by the properties *no junk* and *no confusion*. *No junk*, the prototype contains exactly the elements that must be there, according to the requirements of the specification. *No confusion*, any theorem that holds in the prototype is a consequence of the axioms in the specification, thus holds in every model.
- A sound and complete *deduction calculus*: There is a calculus that allows the derivation of a proposition if and only if this proposition holds in every model.
- *Free functor semantics* for parameterized specifications: The semantics of a parameterized specification $PAR \subseteq BOD$ is given by a functor F that maps PAR–models to BOD–models, such that $F(M)$ is a BOD–prototype relative to the PAR–model M.
- *Compositionality* Specifications can be composed and models of the component specifications can be composed in the same way, such that the composed model is a model of the composed specification.

In general the formulation of such properties in categorical terms induces prerequisites on the structure of specifications or model classes. This observation has led to the development of several abstract theories for specification logics whose purpose is to define minimal requirements or a minimal structure so that properties such as the ones mentioned above can be defined formally.

Syntax and semantics as basic ingredients of such a framework are agreed upon by all approaches in the following sense.

Syntax (Signatures) The syntactic entities, called **signatures**, are organized into a category $\mathcal{SIG}$ with signatures as objects and signature morphisms as morphisms. This allows for instance renaming and syntactic operations for the composition of signatures to be modelled by signature morphisms.

Semantics For each signature Σ in $\mathcal{SIG}$ there is a **category** $Mod(\Sigma)$ **of models**. Thus different models of a signature can be compared via $Mod(\Sigma)$–morphisms. Moreover, change of signature is reflected by a 'forgetful' construction on models and homomorphisms: for each signature morphism $i : \Sigma \to \Sigma'$ there is a functor $Mod(i) : Mod(\Sigma') \to Mod(\Sigma)$ in opposite direction. Thus $Mod : \mathcal{SIG} \to \mathcal{CAT}^{op}$ is a functor.

The pair $(\mathcal{SIG}, Mod : \mathcal{SIG} \to \mathcal{CAT}^{op})$ constitutes a **specification frame** as introduced in [Ehr89, EG94], where $\mathcal{SIG}$, however, is considered as a category of specifications (called $\mathcal{SPEC}$). That is, signatures plus axioms are considered as basic syntactic objects.

In the definition of an **institution** (see [GB84, GB92, Tar96]) sentences are modelled as a third specification component by a functor $Sen : \mathcal{SIG} \to \mathcal{SET}$. Sentences and models are related by a family of **satisfaction relations** $(\models_\Sigma \subseteq Obj_{Mod(\Sigma)} \times Sen(\Sigma))_{\Sigma \in \mathcal{SIG}}$ that is compatible with translation along signature morphisms. That means, for all $i : \Sigma \to \Sigma'$, $M' \in Obj_{Mod(\Sigma')}$, $s \in Sen(\Sigma)$

$$M' \models_{\Sigma'} Sen(i)(s) \Leftrightarrow Mod(i)(M') \models_\Sigma s \; .$$

Specifications in an institution can be defined as pairs (Σ, AX), given by a signature Σ and a subset of sentences $AX \subseteq Sen(\Sigma)$. The corresponding model category $Mod(\Sigma, AX)$ is the subcategory of all Σ–models that satisfy AX. Together with axiom preserving signature morphisms this yields a category $\mathcal{SPEC}$ of specifications.

Further variants of specification logics introduce entailment (see [FS87]), proof systems and combinations thereof (see [Mes89]) as constituent components.

In each of these logics the concepts of prototypes, free functor semantics and compositionality can be formulated as follows.

Prototypes For each specification $SPEC$ the model category $Mod(SPEC)$ has an initial object. (For a justification of this correspondence see [GTW78] and [MG85].)

Free functor semantics For each specification morphism $i : SPEC \to SPEC'$ the forgetful functor $Mod(i) : Mod(SPEC') \to Mod(SPEC)$ has a left adjoint $F_i : Mod(SPEC) \to Mod(SPEC')$.

Compositionality The category $\mathcal{SPEC}$ of specifications has colimits and the model functor $Mod : \mathcal{SPEC} \to \mathcal{CAT}^{op}$ preserves colimits, that is, colimits in $\mathcal{SPEC}$ are mapped to limits in $\mathcal{CAT}$.

Consequences of these properties for a specification logic, in particular concerning parameterization and horizontal and vertical structuring mechanisms, are discussed for instance in [GB85, GB92, EG94].

To state the existence of a sound and complete deduction calculus it is necessary to have a 'logical' component in the framework that permits statements about derivation or entailment, as in [FS87] or MESEGUER'S logical systems [Mes89].

An active research topic is to investigate the morphisms between such specification logics, that would allow us to change between different languages, different types of models, calculi, tools, etc. A discussion of these developments, however, would go beyond the scope of this paper. Probably the most elaborate approach at the moment is [CM93].

Concerning *problem 1* of Section 1 for single specification logics, however, we have shown how the basic notions are related and how they are modelled categorically.

3 Algebras as Functors: the Idea of Classifying Categories

In the following two sections we will sketch the development of specification logics for partial algebras and concurrent systems respectively, using the categorical approach of classifying categories. To explain this approach we present in this section the classifying categories for the most basic case of equational specifications of total algebras. (A survey on classifying categories in first order categorical logic can be found in [KR77], a more algebraic view is presented in [Poi85]. The idea of classifying categories goes back to the fundamental work of LAWVERE, [Law63]. For further applications see the remarks at the end of this section.)

The approach of classifying categories is based on the observation that in most cases specification logics are strongly related to certain categorical structures. In the case of equational algebraic specifications, for instance, the corresponding categorical structure is given by finite products.

We want to denote by $\mathcal{STRUCT}$ the category of all categories with the corresponding categorical structure and with functors preserving this structure. For a given specification *SPEC* we are looking for a classifying category $Cl(SPEC)$, that is, a category in $\mathcal{STRUCT}$ with a universal property that establishes an equivalence between the category of $SPEC$–models and a category of structure preserving functors

$$\textbf{(Cl)} \qquad Mod(SPEC) \simeq Funct_{\mathcal{STRUCT}}[Cl(SPEC), \mathcal{SET}].$$

$Funct_{\mathcal{STRUCT}}[Cl(SPEC), \mathcal{SET}]$ is the *functor category* with objects the $\mathcal{STRUCT}$-preserving functors form $Cl(SPEC)$ to $\mathcal{SET}$ and morphisms the natural transformations between such functors. As usual the universal property determines a classifying category up to isomorphism.

More generally, the purpose of classifying categories is to fix for a given specification logic a categorical structure $\mathcal{STRUCT}$, such that

- each specification $SPEC$ can be considered as a presentation of a category $Cl(SPEC)$ with the structure $\mathcal{STRUCT}$,
- the construction of the classifying category $Cl(SPEC)$ according to the categorical rules specifying the universal properties of the structure $\mathcal{STRUCT}$ induces a sound deduction calculus,

- models are presentations of structure preserving functors from the classifying category to a category with (at least) the same structure, usually $\mathcal{SET}$, homomorphisms are presentations of natural transformations of such functors,
- specification morphisms are presentations of structure preserving functors between classifying categories.

In this section we will first construct a specific classifying category $AT(SPEC)$, the *algebraic theory* of some given equational algebraic specification $SPEC$, by syntactic means alone. Then we show how to derive the most important results of algebraic specifications from the equivalence (Cl), namely

- the existence of initial algebras,
- the recursion theorem,
- a sound and complete deduction calculus,
- free functor semantics for parameterized specifications,
- compositionality.

To achieve the right level of abstraction, that makes it possible to apply this approach to the cases of partial algebras and concurrent systems in the following sections, we give then the general characterization of a classifying category.

The algebraic theory $AT(SPEC)$ of a given specification $SPEC = (S, OP, E)$ can be defined directly. Its objects are strings of sort names and its morphisms are congruence classes of tuples of terms. To make things a bit easier we consider terms as well as equations always with a local variable declaration whose set of variables is an initial segment of $X = \{x_i \mid i \in \mathbb{N}\}$. Then a string of sortnames $w = s_1 \dots s_n$ represents the variable declaration $x_1 : s_1, \dots, x_n : s_n$. This representation yields the source w of a term $t \in T_{OP,s}(\{x_1 : s_1, \dots, x_n : s_n\})$ considered as morphism, its target is s. Thus

- $Obj_{AT(SPEC)} = S^*$
- $Mor_{AT(SPEC)}(w, v)$
  $=\{\ [\langle t_1, \dots, t_k\rangle] \mid t_j \in T_{OP,s'_j}(\{x_1 : s_1, \dots, x_n : s_n\})\ \}$
  where $w = s_1 \dots s_n$, $v = s'_1 \dots s'_k$, and $[\ ]$ refers to the congruence relation generated by E.

Composition is substitution

$$[\langle t_1, \dots, t_k\rangle] \circ [\langle r_1, \dots, r_n\rangle] = [\langle t_1[r_1/x_1, \dots, r_n/x_n], \dots, t_k[r_1/x_1, \dots, r_n/x_n]\rangle]$$

and the identity id_w on $w \in S^*$ is given by the tuple of variables

$$id_w = [x_w] = [\langle x_1, \dots, x_n\rangle]$$

$AT(SPEC)$ has finite products: The product of two objects $w, v \in S^*$ is their concatenation $wv \in S^*$, the projections are corresponding tuples of variables $\pi_1 = [\langle x_1, \dots, x_n\rangle] : wv \to w$ and $\pi_2 = [\langle x_{n+1}, \dots, x_{n+k}\rangle] : wv \to v$ if $w = s_1 \dots s_n$ and $v = s'_1 \dots s'_k$. Tupling (the mediating morphism) of $[\langle t_1, \dots, t_n\rangle] : u \to w$ and $[\langle t'_1, \dots, t'_k\rangle] : u \to v$ is given by $[\langle t_1, \dots, t_n, t'_1, \dots, t'_k\rangle] : u \to wv$.

The terminal object is the empty word $\lambda \in S^*$ with the unique morphisms $[\langle\rangle] : w \to \lambda$ for all $w \in S^*$.

Using the inductive definitions of strings, terms, and the congruence on terms generated by equations it can be shown that each *SPEC*–algebra

$$A = ((A_s)_{s \in S}, (A_{op})_{op \in OP})$$

extends to a finite product preserving functor

$$\tilde{A} : AT(SPEC) \to \mathcal{SET}$$

via

$$\tilde{A}(s_1 \dots s_n) = A_{s_1} \times \cdots \times A_{s_n}$$

$$\tilde{A}([t] : w \to v) = A_t : A_w \to A_v \ \ , \ \ \text{the term function associated with } t.$$

Vice versa each finite product preserving functor $F : AT(SPEC) \to \mathcal{SET}$ can be restricted to a $SPEC$–algebra $\hat{F} = ((F(s))_{s \in S}, (F([op(x_w)])_{op \in OP}))$. These two mappings extend to functors and establish the equivalence

$$Mod(SPEC) \simeq Funct_{\mathcal{PROD}}[AT(SPEC), \mathcal{SET}]$$

where $\mathcal{PROD}$ denotes the category of all categories with finite products and functors preserving these products.

This equivalence can be used to derive the properties of $Mod(SPEC)$ stated in the beginning of this section. The basic categorical result that we need for that purpose is the Yoneda Lemma.

Lemma (Yoneda *) Given a category $\mathcal{C}$ and a functor $F : \mathcal{C} \to \mathcal{SET}$ there is for each object $A \in Obj_{\mathcal{C}}$ a natural bijection*

$$\varphi_A : Nat(h_A, F) \cong F(A) \quad \textit{given by} \quad \varphi_A(\alpha) = \alpha_A(id_A)$$

where h_A denotes the hom-functor $Mor_{\mathcal{C}}(A, \_) : \mathcal{C} \to \mathcal{SET}$ and $Nat(h_A, F)$ denotes the set of natural transformations $\alpha : h_A \Rightarrow F$ from h_A to F. Natural *means that the bijections φ_A are compatible with $\mathcal{C}$–morphisms $f : A \to B$.*

Now we can prove the statements given above.

Initial Algebras For each specification $SPEC$ there is an initial algebra I_{SPEC} in $Mod(SPEC)$.
Proof: Hom-functors preserve products thus the hom–functor h_λ at the terminal object λ of $AT(SPEC)$, $h_\lambda = Mor_{AT(SPEC)}(\lambda, \_) : AT(SPEC) \to \mathcal{SET}$, induces a $SPEC$–algebra $I = \hat{h}_\lambda$. Then the equivalence of model and functor categories and the Yoneda lemma imply

$$Mor_{Mod(SPEC)}(I, A) \cong Nat(h_\lambda, \tilde{A}) \cong \tilde{A}(\lambda) \cong \{*\}$$

the latter since $\tilde{A}$ preserves finite products. Recall that terminal objects are empty products. In this way there is exactly one $SPEC$–homomorphism from I to A for each $SPEC$–algebra A.

Recursion Theorem Given an algebraic specification $SPEC = (S, OP, E)$ and a tuple of variables $x_w = \langle x_1, \ldots, x_n\rangle$ with $x_i : s_i \in S$ there is a $SPEC$–algebra $T(x_w)$ such that for each variable assignment $x_w \mapsto a_w = \langle a_1, \ldots a_n\rangle \in A_{s_1} \times \cdots \times A_{s_n} = A_w$ there is a unique $SPEC$–homomorphism $\bar{a}_w : T(x_w) \to A$ with $\bar{a}_w(x_w) = a_w$.
Proof: As above, with $T(x_w) = \hat{h}_w$ instead of $\hat{h}_\lambda$:

$$\begin{aligned} & Mor_{Mod(SPEC)}(T(x_w), A) \\ \cong\ & Nat(h_w, \tilde{A}) \\ \cong\ & \tilde{A}(w) \\ \cong\ & \tilde{A}(s_1) \times \cdots \times \tilde{A}(s_n) \\ \cong\ & A_{s_1} \times \cdots \times A_{s_n} \end{aligned}$$

Now for $a_w \in A_{s_1} \times \cdots \times A_{s_n}$ we obtain $\bar{a}_w = \varphi^{-1}(a_w)$ and $\bar{a}_w(x_w) = \varphi(\bar{a}_w) = a_w$.

Deduction Calculus Let the deduction of equations $\vdash_{SPEC}$ be defined by

$$\vdash_{SPEC} (w : t = t') \text{ iff } [t] = [t'] : w \to v \text{ in } AT(SPEC).$$

Soundness: If an equation $(w : t = t')$ is derivable with $\vdash_{SPEC}$, then it holds in every $SPEC$–algebra.
Proof: Since for any $SPEC$–algebra A and any term t we have $A_t = \tilde{A}([t])$, $[t] = [t']$ implies $A_t = A_{t'}$.
Completeness: If a ground equation $(\lambda : t = t')$ holds in every $SPEC$–algebra, then it is derivable with $\vdash_{SPEC}$.
Proof: $(\lambda : t = t')$ holds in particular in the initial algebra $I = \hat{h}_\lambda$. Thus

$$[t] = h_\lambda([t])([\langle\rangle]) = I_t = I_{t'} = h_\lambda([t'])([\langle\rangle]) = [t']$$

and by definition $\vdash_{SPEC} (\lambda : t = t')$.

To obtain free functor semantics for parameterized specifications and the compositionality results first observe that a (slightly generalized notion of) specification morphism $i : SPEC \to SPEC'$ can be defined as the restriction of a finite product preserving functor $\tilde{i} : AT(SPEC) \to AT(SPEC')$ to $SPEC$, that is $i = ((i(s))_{s\in S}, (i(op))_{op\in OP})$ with $i(s) \in (S')^*$ and $i(op) \in T_{OP', i(s)}(x_{i^*(w)})$ if $op \in OP_{w,s}$. Then the forgetful functors $V_i : Mod(SPEC') \to Mod(SPEC)$ are defined by $(V_i(B))_s = B_{i(s)}, (V_i(B))_{op} = B_{i(op)}$, and $(V_i(h))_s = h_{i(s)}$ for each $SPEC'$–algebra B and $SPEC'$–homomorphism h. Thus, in the equivalent functor category $Funct_{\mathcal{PROD}}[AT(SPEC), \mathcal{SET}]$, $\widetilde{V_i(B)} = \tilde{B} \circ \tilde{i}$, that is, the forgetful functors are given by composition.

Free Functor Semantics The forgetful functor $V_i : Mod(SPEC') \to Mod(SPEC)$ induced by a specification morphism $i : SPEC \to SPEC'$ has a left adjoint $F_i : Mod(SPEC) \to Mod(SPEC')$.
Proof: First let $A \in Obj_{Mod(SPEC)}$ be a finitely generated algebra, that is $A \cong \hat{h}_w$ for some $w \in S^*$. We show that $F_i(A) := \hat{h}_{\tilde{i}(w)}$ is a free construction.

$$\begin{aligned}
& Mor_{Mod(SPEC')}(F_i(A), B) \\
\cong\; & Nat(h_{\tilde{i}(w)}, \tilde{B}) \\
\cong\; & \tilde{B}(i(w)) \\
\cong\; & Nat(h_w, \tilde{B} \circ \tilde{i}) \\
\cong\; & Mor_{Mod(SPEC)}(A, V_i(B))
\end{aligned}$$

The argument carries over to all of $Mod(SPEC)$ since an arbitrary $SPEC'$–algebra is a filtered colimit of finitely generated $SPEC'$–algebras (A is the union of its finitely generated subalgebras), hom–functors map colimits in their first argument to limits, and filtered colimits of finite product preserving functors preserve finite products.

Compositionality The model functor maps colimits to limits,

$$Mod(Colim_j\, SPEC_j) \cong Lim_j\, Mod(SPEC_j)$$

for each diagram of specifications $(SPEC_j)_{j \in J}$.
Proof: First it has to be shown that the category of specifications and specification morphisms is cocomplete. This can either be done directly, or using the fact that the category of specifications is itself essentially algebraic. Since the algebraic theories are freely generated by the specifications we have furthermore that $AT(Colim_j\, SPEC_j) \simeq Colim_j\, AT(SPEC_j)$. To show that the model functor $Mod : \mathcal{SPEC} \to \mathcal{CAT}^{op}$ maps these colimits to limits in $\mathcal{CAT}$ consider first the isomorphism of the full functor categories

$$\begin{aligned}
& Funct_{\mathcal{CAT}}[Colim_j\, AT(SPEC_j), \mathcal{SET}] \\
\simeq\; & Lim_j\, Funct_{\mathcal{CAT}}[AT(SPEC_j), \mathcal{SET}]
\end{aligned}$$

given by

$$\begin{array}{lll}
f : Colim_j\, AT(SPEC_j) \to \mathcal{SET} & \mapsto & (f \circ in_j : AT(SPEC_j) \to \mathcal{SET})_{j \in J} \\
(g_j : AT(SPEC_j) \to \mathcal{SET})_{j \in J} & \mapsto & g : Colim_j\, AT(SPEC_j)_{j \in J} \to \mathcal{SET}
\end{array}$$

where $in_j : AT(SPEC_j) \to Colim_j\, AT(SPEC_j)$ is the colimit injection and $g : Colim_j\, AT(SPEC_j) \to \mathcal{SET}$ is the mediating morphism of the compatible family $(g_j : AT(SPEC_j) \to \mathcal{SET})_{j \in J}$. Now the isomorphism above restricts to the subcategories $\mathcal{PROD}$ of finite product preserving functors, since $\mathcal{PROD}$ has finite products. Thus

$$\begin{aligned}
& Mod(Colim_j\, SPEC_j) \\
\simeq\; & Funct_{\mathcal{PROD}}[AT(Colim_j\, SPEC_j), \mathcal{SET}] \\
\simeq\; & Funct_{\mathcal{PROD}}[Colim_j\, AT(SPEC_j), \mathcal{SET}] \\
\simeq\; & Lim_j\, Funct_{\mathcal{PROD}}[AT(SPEC_j), \mathcal{SET}] \\
\simeq\; & Lim_j\, Mod(SPEC_j)
\end{aligned}$$

Let's sum up the results for $\mathcal{SET}$–models for the general approach as

- If the classifying category has a terminal object 1 and the hom–functor h_1 is structure preserving, then $Mod(SPEC)$ has an initial model, and the deduction calculus is sound and complete.
- If the classifying category has finite products and all hom–functors are structure preserving, then the recursion theorem holds.
- If furthermore the class of structure preserving functors coincides with the class of filtered colimits of hom–functors, then we have free functor semantics for specification morphisms.
- If the category of specifications is cocomplete and the isomorphism of the full functor categories defining limits and colimits restricts to the subcategories of structure preserving functors, then we have compositionality.

The fact that categories with finite limits and finite limit preserving functors have these properties will be used in the following section to establish these results for a specification logic for partial algebras.

A disadvantage of the concrete syntactic definition of algebraic theories as classifying categories as discussed up to now is, that they depend on the predefined notions of strings, terms, substitution, and congruence generated by equations. If the approach shall be applied to develop a specification logic — as in the following two sections – it is necessary, however, to formulate the essential structure before the syntax of the specification logic is fixed. To obtain such a syntax independent definition of a classifying category we have to make its universal property more precise.

First we make precise the relation between an algebraic specification $SPEC = (S, OP, E)$ and an arbitrary classifying category $Cl(SPEC)$ of $SPEC$, and indicate this relation for the specific classifying category $AT(SPEC)$. $Cl(SPEC)$ has to be a category with finite products, such that:

- For each sort $s \in S$ there is an object ob_s in $Cl(SPEC)$.
  (In $AT(SPEC)$ $ob_s = s \in S^*$.)
- For each operation symbol $op : w \to s \in OP$
  there is a morphism $m_{op} : ob_w \to ob_s$ in $Cl(SPEC)$,
  where $ob_w = ob_{s_1} \times \cdots \times ob_{s_n}$ if $w = s_1 \ldots s_n$.
  (In $AT(SPEC)$ $m_{op} : ob_w \to ob_s = [\langle op(x_w)\rangle] : w \to s$.)
- For each equation $(w\!:\!t = t') \in E$
  the associated morphisms $m_t, m_{t'} : ob_w \to ob_s$ in $Cl(SPEC)$ are equal,
  where the morphisms $m_t : ob_{s_1} \times \cdots \times ob_{s_n} \to ob_s$
  for terms $t \in T_{OP,s}(\{x_1 : s_1, \ldots, x_n : s_n\})$ are defined as follows.
  - variables, $t = x_i$:
    $m_{x_i} = \pi_i : ob_{s_1} \times \cdots \times ob_{s_n} \to ob_{s_i}$
    the i'th projection.
  - tuples, $t = \langle t_1, \ldots, t_k\rangle$:
    $m_t = \langle m_{t_1}, \ldots, m_{t_k}\rangle$
    where the brackets $\langle , \rangle$ denote the tupling corresponding to the product structure of $Cl(SPEC)$

- composed terms, $t = op(r)$, $op : w \to s \in OP$:

  $$m_{op(r)} = m_{op} \circ m_r$$

  (In $AT(SPEC)$ $m_t = [\langle t \rangle]$, thus for $(w : t = t') \in E$, $m_t = [\langle t \rangle] = [\langle t' \rangle] = m_{t'}$.)

Note that replacing $Cl(SPEC)$ in this definition by $\mathcal{SET}$ we obtain the usual definition of a $SPEC$–algebra. The generalization of $SPEC$–algebras to $SPEC$–algebras in an arbitrary category $\mathcal{C}$ with finite products is then straight forward: just replace $Cl(SPEC)$ in the definition above by $\mathcal{C}$. Accordingly, a $SPEC$–homomorphism $h : A^{(1)} \to A^{(2)}$ in $\mathcal{C}$ is a family $(h_s : A_s^{(1)} \to A_s^{(2)})_{s \in S}$ of morphisms in $\mathcal{C}$ with $h_s \circ A_{op}^{(1)} = A_{op}^{(2)} \circ (h_{s_1} \times \cdots \times h_{s_n})$ for all $op : s_1 \ldots s_n \to s \in OP$. This yields for each category $\mathcal{C}$ with finite products a category $Mod_{\mathcal{C}}(SPEC)$ of $SPEC$–algebras and $SPEC$–homomorphisms in $\mathcal{C}$. (Thus $Mod(SPEC) = Mod_{\mathcal{SET}}(SPEC)$.) The relation between $SPEC$ and $Cl(SPEC)$ can now be reformulated as:

$$G := ((ob_s)_{s \in S}, (m_{op})_{op \in OP}) \text{ is a } SPEC\text{–algebra in } Cl(SPEC)$$

(called the **generic algebra**) and the universal property of a classifying category $Cl(SPEC)$ together with the generic algebra G in $Cl(SPEC)$ is:

> For each $SPEC$–algebra $A = ((A_s)_{s \in S}, (A_{op})_{op \in OP})$ in some category $\mathcal{C}$ there is a functor $\tilde{A} : Cl(SPEC) \to \mathcal{C}$, unique up to natural isomorphism, with $\tilde{A}(ob_s) = A_s$ for all $s \in S$ and $\tilde{A}(m_{op}) = A_{op}$ for all $op \in OP$.

(Thus each algebra is an image of the generic algebra.) The universal property implies that the restriction mapping

$$(P : Cl(SPEC) \to \mathcal{C}) \mapsto (\hat{P} = ((P(ob_s))_{s \in S}, (P(m_{op}))_{op \in OP}))$$

is an inverse (up to isomorphism) to the extension $A \mapsto \tilde{A}$, that is,

$$\tilde{\_} : Mod_{\mathcal{C}}(SPEC) \to Funct_{\mathcal{PROD}}[Cl(SPEC), \mathcal{C}]$$

is the desired equivalence. (As above, both mappings $\tilde{\_}$ and $\hat{\_}$ extend to functors.)

Classifying categories have been developed for several type theoretic and logical frameworks. A correspondence between locally cartesian closed categories and (a variant of) Martin–Löf type theory is set up in [See84]. Cartesian closedness as categorical infrastructure for (typed) λ–calculus and topoi for higher order intuitionistic logic are presented in [LS86]. [See87] and [HP89] present categorical semantics for higher order polymorphic lambda calculus and the theory of constructions.

Interaction categories (see [Abr93, Abr94]) as categorical models of concurrency are developed similarly, starting from $*$–autonomous categories as representation of linear logic.

4 The Development of a Specification Logic for Partial Algebras

As solution to *problem 2* stated in Section 1 we now apply the approach of classifying categories discussed above to the specification of partial algebras. A very detailed presentation of the development of a specification logic for partial algebras as sketched here can be found in [CGW95]. The standard reference for partial algebras is the book [Rei87], which is based on similar ideas, and the earlier presentations [KR72, Fre72, Fre73].

The first step in the development of a specification logic for partial algebras is straight forward. Signatures and models are defined as for total algebras, except that an operation symbol $op : s_1 \dots s_n \rightarrow s$ is interpreted by a partial function $A_{op} : A_{s_1} \times \cdots \times A_{s_n} \rightharpoonup A_s$ instead of a total one. But for the subsequent notions of axiom and homomorphism several alternatives appear.

- What is the right choice for the axioms? If equations $t = t'$ are used, how is satisfaction defined? Does it mean that if t and t' can be evaluated then they must yield the same result? Or does it mean that t and t' must both be defined and equal? Moreover, do conditional existence equations increase the complexity of the logic?
- What is the right choice for the homomorphisms? Should these be partial or total functions? Should the domains of definition of the operations be preserved or reflected? Should the compatibility property of homomorphisms $h(A_{op}(a_1, \dots, a_n)) = B_{op}(h(a_1), \dots, h(a_n))$ be restricted to the elements both sides are defined for?
  This question has, of course, an impact on the definitions of subalgebra (injective homomorphism, monomorphism), quotient algebra (surjective homomorphism, epimorphism), and initial algebra, as well as the existence and role of term algebras.

As a guideline through these design decisions we now apply the classifying categories approach. Using their universal property we can first develop the right categorical structure, without depending on a predefined syntax. So we start by motivating why finite limits are the appropriate categorical structure to model partial algebras, and then we develop a concrete syntactic classifying category $CT(SPEC)$ for a partial specification $SPEC$, its *cartesian theory*. The desired results follow then from the *guidelines for the development of a specification logic* in Section 3.

The first observation in the development of a classifying category for a partial specification is that a partial operation A_{op} corresponding to an operation symbol $op : w \rightarrow s \in OP$ in a partial algebra A is given by a span $A_w = A_{s_1} \times \cdots \times A_{s_n} \supseteq dom(A_{op}) \xrightarrow{A_{op}} A_s$, where the subset $dom(A_{op})$ is the domain of definition of A_{op} and $dom(A_{op}) \xrightarrow{A_{op}} A_s$ is a total function. Thus in the classifying category we must have spans $ob_w = ob_{s_1} \times \cdots \times ob_{s_n} \xleftarrow{in_{op}} ob_{dom(op)} \xrightarrow{m_{op}} ob_s$, where in_{op} is a monomorphism, to represent the operation symbols $op \in OP$. Since (strict) composition of partial functions f and g is defined by

$$dom(g \circ f) = \{x \in dom(f) \mid f(x) \in dom\, g\} = f^{-1}(dom\, g)$$
$$g \circ f(x) = g(f(x)) \quad (x \in dom(g \circ f))$$

and inverse images are pullbacks, the classifying category should have pullbacks. Suppose a tuple of terms $t = \langle t_1, \ldots, t_n \rangle$ has already been represented as a span

$$ob_v \xleftarrow{in_t} ob_{dom(t)} \xrightarrow{m_t} ob_w$$

we can construct the pullback:

$$\begin{tikzcd}
ob_v & & & ob_w & & & ob_s \\
 & = & ob_{dom(t)} \arrow[ull, "in_t"'] \arrow[ur, "m_t"] & (pb) & ob_{dom(op)} \arrow[ul, "in_{op}"'] \arrow[urr, "m_{op}"] & = & \\
 & & & ob_{dom(op(t))} \arrow[ul] \arrow[ur] \arrow[uulll, "in_{op(t)}"] \arrow[uurrr, "m_{op(t)}"'] & & &
\end{tikzcd}$$

So a composed term $op(t) = op(t_1, \ldots t_n)$ is represented by the span

$$ob_v \xleftarrow{in_{op(t)}} ob_{dom(op(t))} \xrightarrow{m_{op(t)}} ob_s$$

by pullbacks and composition, provided the tuple of terms $t = \langle t_1, \ldots . t_n \rangle$ has already been represented. To represent such a tuple first the common domain $ob_{dom(\langle t_1, \ldots, t_n \rangle)}$ has to be constructed. This is again a (multiple) pullback.

$$\begin{tikzcd}
ob_{u_i} & & & ob_v & & & ob_{u_j} \\
 & = & ob_{dom(t_i)} \arrow[ull, "m_{t_i}"'] \arrow[ur, "in_{t_i}"] & (pb) & ob_{dom(t_j)} \arrow[ul, "in_{t_j}"'] \arrow[urr, "m_{t_j}"] & = & \\
 & & & ob_{dom(\langle \ldots, t_i, t_j, \ldots \rangle)} \arrow[ul] \arrow[ur] \arrow[uulll, "m_{t_i}"] \arrow[uurrr, "m_{t_j}"'] & & &
\end{tikzcd}$$

Then tupling yields

$$m_{\langle t_1, \ldots, t_n \rangle} = \langle m_{t_1}, \ldots, m_{t_n} \rangle : ob_{\langle t_1, \ldots, t_n \rangle} \to ob_{u_1} \times \cdots \times ob_{u_n} \quad .$$

Finite products together with pullbacks yield all finite limits, in particular equalizers. In $\mathcal{SET}$ equalizers are solution sets

$$Eq(f, g : A \to B) = \{x \mid f(x) = g(x)\} \subseteq A\,.$$

Thus in $Cl(SPEC)$ we have for each pair of spans $ob_w \xleftarrow{in_t} ob_{dom(t)} \xrightarrow{m_t} ob_s$, $ob_w \xleftarrow{in_{t'}} ob_{dom(t')} \xrightarrow{m_{t'}} ob_s$, given by the terms $t, t' \in T_{OP,s}(\{x_1 : s_1, \ldots, x_n : s_n\})$ an equalizer

$$\begin{tikzcd} ob_{(w:t=t')} \arrow[r] \arrow[rr, bend right, "m_t = m_{t'}"'] & ob_{dom(\langle t,t'\rangle)} \arrow[r, shift left, "m_t"] \arrow[r, shift right, "m_{t'}"'] & ob_s \end{tikzcd}$$

where $ob_{dom(\langle t,t'\rangle)}$ is the pullback of the domains of t and t' respectively. Note that the equalizer is contained in the intersection of the domains of definition of t and t', thus equations are existence equations: $t = t'$ implies both t and t' are defined. In particular $t = t$ means that t is defined.

Now we have objects $ob_{(w:t=t')}$ representing (existence) equations, whose interpretation in $\mathcal{SET}$ (by a finite limit preserving functor) are solution sets, and we have monomorphisms. Thus conditional existence equations $(w : r = r' \to t = t')$, represented by monomorphisms $ob_{(w:r=r')} \to ob_{(w:t=t')}$, are the natural choice as axioms, since they represent the inclusion of the solution sets.

Now we have all the information to define the 'natural' (categorical) versions of specification, partial algebra, homomorphism, specification morphism and forgetful functor. And we already know — according to the *guidelines* in Section 3 — that the so defined specification logic has initial models, the recursion theorem holds, there is a sound and complete deduction calculus, free functor semantics for parameterized specifications, and compositionality, since we use classifying categories with finite limits.

Firstly, a partial specification $SPEC = (\Sigma, CE)$ is defined as a pair that consists of an algebraic signature $\Sigma = (S, OP)$ and a set of conditional equations CE (interpreted as conditional existence equations).

Before we come to the other definitions we construct a concrete classifying category, the *cartesian theory* $CT(SPEC)$, as a category with finite limits, and the generic partial algebra

$$G = ((ob_s)_{s \in S}, (in_{op}, ob_{dom(op)}, m_{op})_{op \in OP})$$

in $CT(SPEC)$. For that purpose we can use again the known syntactic notions of strings of sort names, terms, substitution and equations. Objects are equations (with variable declarations represented by strings of sortnames) and morphisms are equivalence classes of tuples of terms, which are constructed together with their source and target as follows.

- For each sort $s \in S$ introduce an object $ob_s = (s : x_s = x_s)$.
- Construct finite products
  $ob_{s_1} \times \ldots \times ob_{s_n} = ob_w = (w : x_w = x_w)$,
  including the terminal object $(\lambda : x_\lambda = x_\lambda)$ with projections
  $\pi_i = [x_i] : (w : x_w = x_w) \to (s_i : x_{s_i} = x_{s_i})$.
- For each operation symbol $op : w \to s \in OP$ introduce its span, that is, an object

$ob_{dom(op)} = (w : op(x_w) = op(x_w))$
and morphisms
$in_{op} = [x_w] : (w : op(x_w) = op(x_w)) \to (w : x_w = x_w)$
$m_{op} = [op(x_w)] : (w : op(x_w) = op(x_w)) \to (s : x_s = x_s)$.

– Construct recursively
  - composition of morphisms
    $[t] : (v : r_1 = r_2) \to (w : t_1 = t_2)$
    $[r] : (w : t_1 = t_2) \to (u : z_1 = z_2)$
    by simultaneous substitution of terms
    $[r] \circ [t] = [r[t/x_w]](v : r_1 = r_2) \to (u : z_1 = z_2)$
  - tuples
    $[\langle t_1, \ldots, t_n \rangle] : (u : r_1 = r_2) \to (w : x_w = x_w)$
    of morphisms $[t_i] : (u : r_1 = r_2) \to (s_i : x_{s_i} = x_{s_i})$
  - pullbacks of monos
    $[x_w] : (w : t_1 = t_2) \to (w : x_w = x_w)$
    and morphisms
    $[t] : (v : r_1 = r_2) \to (w : x_w = x_w)$
    by
    $[x_v] : (v : r_1 = r_2 \wedge t_1[t/x_w] = t_2[t/x_w]) \to (v : r_1 = r_2)$
    and
    $[t] : (v : r_1 = r_2 \wedge t_1[t/x_w] = t_2[t/x_w]) \to (w : t_1 = t_2)$
  - equalizers of
    $[t], [t'] : (v : r_1 = r_2) \to (w : x_w = x_w)$
    by
    $[x_v] : (v : r_1 = r_2 \wedge t = t') \to (v : r_1 = r_2)$
    with
    $[t] = [t'] : (v : r_1 = r_2 \wedge t = t') \to (w : x_w = x_w)$
  - and introduce for each conditional equation $(w : r = r' \to t = t') \in CE$ a monomorphism $[x_w] : (w : r = r') \to (w : t = t')$.

The so defined category $CT(SPEC)$ has finite limits (by construction) and classifies the following model category $PAlg(SPEC)$. (We only consider $\mathcal{SET}$–models here. To obtain partial algebras in an arbitrary category with finite limits just replace 'sets' by 'objects', 'functions' by 'morphisms', and 'inclusions' by 'monos'.) Recall that the models and homomorphisms are restrictions of structure preserving functors and natural transformations to the generic model $G = ((s : x_s = x_s)_{s \in S}, ([x_w], (w : op(x_w) = op(x_w)), [op(x_w)])_{op \in OP})$ in $CT(SPEC)$.
Let $SPEC = (S, OP, CE)$. A *partial SPEC–algebra A* is given by

– a set A_s for each sort $s \in S$,
  corresponding to the objects $(s : x_s = x_s)$
– a span $A_w \supseteq dom(A_{op}) \xrightarrow{A_{op}} A_s$ for each operation symbol $op : w \to s \in OP$,
  corresponding to the spans
  $(w : x_w = x_w) \xleftarrow{[x_w]} (w : op(x_w) = op(x_w)) \xrightarrow{[op(x_w)]} (s : x_s = x_s)$

– such that $A_{(w:r=r')} \subseteq A_{(w:t=t')}$ for each conditional equation $(w : r = r' \to t = t') \in CE$,
where terms t and equations $(w : t = t')$ are interpreted in A as follows.
  - variables, $t = x_i \in T_{OP,s_i}(\{x_1 : s_1, \ldots, x_n : s_n\})$
    $dom(A_{x_i}) = A_{s_1} \times \cdots \times A_{s_n}$
    $A_{x_i} = \pi_i$
    the i'th projection.
  - tuples, $t = \langle t_1, \ldots, t_k\rangle \in T_{OP,v}(\{x_1 : s_1, \ldots, x_n : s_n\})$
    $dom(A_t) = \bigcap_{i=1,\ldots k} dom(A_{t_i})$
    $A_t = \langle A_{t_1}, \ldots, A_{t_k}\rangle$
  - composed terms, $t = op(r) \in T_{OP,s}(\{x_1 : s_1, \ldots, x_n : s_n\})$, $op : v \to s \in OP$
    $dom(A_{op(r)}) = \{a \in dom(A_r) \mid A_r(a) \in dom(A_{op})\}$
    $A_{op(r)} = A_{op} \circ A_r$
  - $A_{(w:t=t')} = \{a_w \in dom(A_t) \cap dom(A_{t'}) \mid A_t(a_w) = A_{t'}(a_w)\}$,
    the equalizer of A_t and $A_{t'}$.

corresponding to the construction of terms and solution sets via pullbacks, composition and equalizers.

A *SPEC–homomorphism* $h : A \to B$ is given by

– a total function $h_s : A_s \to B_s$ for each sort $s \in S$,
the components corresponding to the objects $(s : x_s = x_s)$,
– such that the domains of definition are preserved,
$a \in dom(A_{op}) \Rightarrow h(a) \in dom(B_{op})$,
the components $h_{(w:op(x_w)=op(x_w))} : A_{(w:op(x_w)=op(x_w))} \to B_{(w:op(x_w)=op(x_w))}$,
– and the operations are preserved,
$$h_s(A_{op}(a_1, \ldots, a_n)) = B_{op}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)),$$
corresponding to the naturality condition.

The deduction calculus for conditional existence equations is given by

$\vdash_{SPEC} (w : r = r' \to t = t')$ iff
there is a subobject inclusion $(w : r = r') \xrightarrow{[x_w]} (w : t = t')$ in $CT(SPEC)$.

According to the construction of $CT(SPEC)$ as a category with finite limits this shows that the categorical definitions (of composition, products, equalizers etc.) are the rules of the calculus. This calculus is sound and complete w.r.t. the above model category. Free functor semantics for parameterized specifications and compositionality are obtained as above, according to the *guidelines* in Section 3. Thus we have obtained a specification logics for partial algebras with the same basic properties as the one of total algebras, as required in *problem 2* in Section 1.

5 Rewrite Logic as a General Framework for Concurrency

We now come to the discussion of *problem 3*.

One of the most prominent challenges to algebraic specification in the recent years has been the specification of concurrent systems. Although at a first glance algebras with their stateless set theoretic functions do not seem to be appropriate as a model for concurrent systems an approach has been presented in [Mes90, Mes92] that — with a slight alteration in the basic definitions — manages to capture a wide variety of models for concurrency. This approach can be presented in terms of classifying categories as discussed in Section 3, where the development of the approach shows up most clearly.

Recall that in the classifying category of a usual algebraic specification morphisms are congruence classes of terms, where the congruence is given by the equations of the specification and a derivation calculus. Now these congruence classes can be refined so that computation within a congruence class is made explicit. (Compare the computation of normal forms which gives an operational flavour to algebraic specifications.) So to model concurrency syntactic equations are considered not to identify terms, but specify explicit rewritings. To allow a greater flexibility than pure rewriting, however, equations are subdivided into structural equations (as, for example, associativity of constructors) that are interpreted identically as before, and proper rewriting rules that are used to specify computation steps.

Now the classifying category approach helps to develop deduction and model theory, where the extension by directed equations as rewrite rules has to be taken into account. The categorical notion for this extension is that of a 2–category that introduces 'morphisms between morphisms'. A 2–category is a category that has *categories of morphisms* instead of sets, and composition is a class of functors $\circ : Mor(A,B) \times Mor(B,C) \to Mor(A,C)$. The *second level* morphisms, that is, the morphisms of the morphism categories $Mor(A,B)$ are called *2–cells* (see [Kel74]).

To carry this through consider the following example of an object oriented concurrent system. A state of this system is given by a multiset of object states and messages

$$\{\!|\langle O_1 : C_1 \mid attr_1\rangle, \ldots, \langle O_n : C_n \mid attr_n\rangle, msg_1, \ldots, msg_k|\!\}$$

where an object state consists of the object identifyer O_i, its class name C_i, and its attributes $attr_i = \langle a_{i,1} : v_{i,1}, \ldots, a_{i,m_i} : v_{i,m_i}\rangle$.

A computation step in this system is given by the replacement of some object states and messages in the given system state, as indicated by the rule

$$\begin{aligned}
& msg_1, \ldots, msg_j, \langle O_1 : C_1 \mid attr_1\rangle, \ldots, \langle O_m : C_m \mid attr_m\rangle, \\
\Rightarrow\ & \langle O_{i_1} : C'_{i_1} \mid attr'_{i_1}\rangle, \ldots, \langle O_{i_k} : C'_{i_k} \mid attr'_{i_k}\rangle, \\
& \langle Q_1 : D_1 \mid attr''_1\rangle, \ldots, \langle Q_l : D_l \mid attr''_l\rangle, \\
& msg'_1, \ldots, msg'_q
\end{aligned}$$

where $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$. The intended meaning of this rule is that

– the messages $msg_1, \ldots, msg_j$ disappear,
– the attributes and possibly even the class of the objects $O_{i_1}, \ldots, O_{i_k}$ change,
– all other objects O_j vanish
– new objects $Q_1, \ldots, Q_l$ are created,
– new messages $msg'_1, \ldots . msg'_q$ are sent.

Different types of replacement rules correspond to different types of concurrent behaviour. For example, the appearance of several objects on the left side of the replacement rule models synchronous computation steps, whereas rules with a single object on their left hand side model asynchronous steps.

Let's now specify this system. In general an $\mathcal{R}$–**specification** (rewrite specification) is given by

– an algebraic signature $\Sigma = (S, OP)$,
– a set E of structural equations $e = (w : t_1 = t_2)$,
– a set R of rewrite rules $r = (w : t_1 \Rightarrow t_2)$.

Suppose an algebraic specification of object identifiers, class names, messages and attributes is already given. It remains to specify multisets as static algebraic structure that represents the states.

```
type state =
  sorts state
  opns  ⟨_ : _|_⟩ : object-id class-name attribute → state
        _ : message → state
        _,_ : state state → state [assoc, comm, id:∅]
end type
```

The attribute [*assoc, comm, id:*$\emptyset$] is a shorthand of the structural equations for associativity, commutativity, and identity. According to this specification rewrite rules have exactly the form given above.

Next we can apply the 2–categorical approach to obtain a classifying category $Cl(\mathcal{R})$ of a specification $\mathcal{R}$ as a 2–category with finite products freely generated by the generic model $G = ((ob_s)_{s \in S}, (m_{op})_{op \in OP}, (c_r)_{r \in R})$ given by

– an object ob_s for each sort $s \in S$,
– a morphism $m_{op} : ob_{s_1} \times \cdots \times ob_{s_n} \to ob_s$ for each operation symbol $op : s_1 \ldots s_n \to s \in OP$
– such that $m_t = m_{t'} : w \to s$ for all $(w : t = t') \in E$, where the morphisms $m_t, m_{t'}$ corresponding to the terms t, t' are obtained by composition and tupling,
– a 2–cell $c_r : m_t \Rightarrow m_{t'}$ for each rewrite rule $r = (w : t \Rightarrow t') \in R$.

As discussed above an $\mathcal{R}$–model M is the restriction to G of a finite product preserving 2–functor from $Cl(\mathcal{R})$ to *the category of categories*. Passing from categories to 2–categories on the syntactical level corresponds to a lifting from the category of sets to the category of categories on the semantic level. That means a model of an $\mathcal{R}$–specification is given by

- a category M_s for each sort $s \in S$,
- a functor $M_{op} : M_w \to M_s$, for each operation symbol $op : w \to s \in OP$ where $M_w = M_{s_1} \times \cdots \times M_{s_n}$ if $w = s_1 \ldots s_n$,
- such that for each structural equation $(w : t = t') \in E$ the associated functors $M_t, M_{t'} : M_w \to M_s$ are equal,
- for each rewrite rule $r = (w : t \Rightarrow t') \in R$ a natural transformation $M_r : M_t \to M_{t'}$

A homomorphism $h : M \to M'$ is then given by

- a family of functors $h_s : M_s \to M'_s$,
- such that $h_s \circ M_{op} = M'_{op} \circ (h_{s_1} \times \cdots \times h_{s_n})$ for each operation $op : s_1 \ldots s_n \to s \in OP$,
- and $h_s \circ M_r = M'_r \circ (h_{s_1} \times \cdots \times h_{s_n})$ for each rewrite rule $r = (w : t \Rightarrow t') \in R$

According to the *guidelines* in Section 3 the structural properties of the so defined specification logic for concurrent systems are the same as for algebraic specifications and the specifications of partial algebras above. That is, there is an initial term model for each $\mathcal{R}$-specification, the recursion theorem holds, forgetful functors have left adjoints and the semantics is compositional with respect to the composition of $\mathcal{R}$–specifications by colimits. These are exactly the properties required in *problem 3* in Section 1.

Concerning the applicability of this approach Meseguer has shown in [Mes92] that the example given above belongs to a family of $\mathcal{R}$–models that contains a variety of models for concurrency. (See the diagram below.) Thus (conditional) rewriting can also be seen as a unifying theory for these different models. The family of models is obtained simply by considering different structural equations for the constructor `_,_ : state state → state`. Concurrent object oriented programming could be modelled using the structural equations for associativity, commutativity, and identity (A, C, I). Other combinations of A, C and I yield the following tree of models (for a detailed discussion see [Mes92].)

$$\begin{tikzcd}
 & & & & CR \arrow[dlll] \arrow[d] \arrow[drr] & & & & & \\
 & CR_{\emptyset} \arrow[dl] \arrow[dr] & & & CR_{AI} \arrow[d] & & CR_{ACI} \arrow[dl] \arrow[d] \arrow[drr] & & & \\
 LbdTransSys & & FunctlProgr \arrow[dl] \arrow[dr] \arrow[dd] & & PostSys \arrow[d] & Petri & Cham \arrow[d] & & ConcOOP \arrow[dl] \arrow[d] \arrow[dr] & \\
 & \lambda\text{-}Calc & & ADT\text{'s} & PhrStrGrammars \arrow[d] & & CCS & Actors & POP & UNITY \\
 & & Herbrand\text{–}Gödel\text{–}Kleene & & Turing & & & & &
\end{tikzcd}$$

Here CR_E denotes the rewriting with the corresponding set of structural equations and the other nodes are:

LbdTransSys	labelled transition systems
FunctlProgr	functional programming
λ–Calc	lambda calculus
Herbrand–Gödel–Kleene	theory of general recursive functions
ADT's	algebraic data types
PostSys	Post Systems
PhrStrGrammars	phrase structure grammars
Turing	Turing machines
Petri	Petri Nets
Cham	chemical abstract machine
CCS	calculus of communicating systems
ConcOOP	concurrent object oriented programming
Actors	actors
POP	parallel object-based systems
UNITY	UNITY, a parallel programming language

6 Conclusion

We have tried to elucidate the role category theory may take in theoretical computer science. Category theory is — in the first place — a way of considering and approaching things. Its basic paradigm, a formal structuralism, makes it much more appropriate for the analysis, description and construction of abstract structures and formal systems than for instance set theory. Being able to describe complex structures with the same language as elementary, even foundational ones, category theory provides one common paradigm that helps in seeing the common form in all the different phenomena. The presentation of the categorical approaches to the three selected problems has shown how category theory can be applied.

It provides *methodological guidelines* for the analysis, structuring and formulation of computer science problems, in that it offers a collection of *basic notions* for the analysis and structuring and a *language* for the formulation. To obtain a general notion of specification logic for instance (see Section 2) syntactical entities (signatures, specifications) are organized *into a category*. Then the semantical entities are associated as an *indexed category*, that is, for each syntactic object there is a *category of models*, and for each morphism of syntactic objects a *forgetful functor* with appropriate compatibility conditions. If further components are added, like sets of sentences, satisfaction, entailment, proof systems, these have to respect the indexed category structure and may introduce further categorical structure.

The same *basic notions and language* can be used to analyse and reformulate a theory in order to *generalize* horizontally or *abstract* vertically. In the first case the essential structure of the theory is separated from its specific details to exhibit its essential relationship with other theories, that only differ in the specific details. The use of classifying categories for total algebras, partial algebras, and rewrite logic (see Sections 3, 4, and 5) demonstrates such a horizontal generalization. In the second case the essential structure of a theory is enlarged to make it more powerful. Taking categories with finite limits instead of finite products as classifying categories allows the modelling of algebraic specifications with conditional equations, and as we have seen also partial operations (and predicates). Taking 2–categories instead of categories allows the modelling of concurrent systems instead of algebras.

The *universal constructions and results* of category theory lead to a consideration of specific elements of a theory or problem once it has been formulated categorically. For example, since models are associated with signatures (in a specification logic) *indexed categories* or *fibrations* are the categorical formulation of syntax and semantics. This implies that signatures must constitute a category, that is, there must be signature morphisms, and there must be forgetful functors. In the case of classifying categories *hom–functors* appear as analogons to free generated (term–) algebras. Then the *Yoneda Lemma* immediately implies the recursion theorem and the existence of initial algebras. Moreover free functors induced by specification morphisms are *Kan extensions*, and the modelling of specifications in this approach shows that specification morphisms are essentially *Kleisli morphisms*.

Let us finally come to the question of how to further apply category theory to problems arising in computer science beyond the examples that we have shown. The first step consists of a *categorical analysis* of the problem field leading to a *categorical formulation*. This modelling includes abstractions and idealizations to make the problem amenable to a mathematical treatment. It may also impose conditions on the problem field that either restrict the applicability of the theory or lead to a restructuring of the problem. As we have shown the categorical notions and language induce a methodological guideline for that purpose.

When the categorical formulation is accomplished the categorical constructions can be used for an investigation of the problem and categorical theorems can be applied to obtain solutions for specific problems.

But still work remains to be done to apply the results to the original problem. In general category theoretic constructions are defined *up to isomorphism* (or by universal properties), thus a concrete representation has to be found. Depending on the problem this might consist, for instance, in the design of a language and/or an appropriate notation. And as third and last step an accessible presentation of the solution of the problem, in terms of the language of the problem, has to be given.

References

[Abr93] S. Abramsky. Interaction categories (extended abstract). In J.L. Burn, S.J. Gay, and M.D. Ryan, editors, *Theory and Formal Methods 1993: Proc. First Imperial College Department of Computing Workshop on Theory and Formal Methods*, pages 57–70. Springer Verlag Workshops in Computer Science, 1993.

[Abr94] S. Abramsky. Interaction categories and communicating sequential processes. In A.W. Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R.Hoare.* Prentice Hall International, 1994.

[BW85] M. Barr and C. Wells. *Toposes, triples, and theories.* Springer Verlag, 1985.

[BW90] M. Barr and C. Wells. *Category Theory for Computing Science.* Prentice Hall, 1990.

[CGW95] I. Claßen, M. Große–Rhode, and U. Wolter. Categorical concepts for parameterized partial specifications. *Math. Struct. in Comp. Science*, 5(2):153–188, 1995.

[CM93] M. Cerioli and J. Meseguer. May I Borrow Your Logic? Technical report, SRI International, Menlo Park, 1993.

[EG94] H. Ehrig and M. Große-Rhode. Functorial theory of parameterized specifications in a general specification framework. *TCS*, (135):221 – 266, 1994.

[Ehr89] H. Ehrig. Algebraic specification of modules and modular software systems within the framework of specification logics. Technical Report 89-17, TU Berlin, 1989.

[EM45] S. Eilenberg and S. MacLane. General theory of natural equivalences. *Trans. Am. Math. Soc.*, 58:231–294, 1945.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science.* Springer, Berlin, 1985.

[EM90] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, volume 21 of *EATCS Monographs on Theoretical Computer Science.* Springer, Berlin, 1990.

[EM95] H. Ehrig and B. Mahr. A decade of TAPSOFT: Aspects of progress and prospects in theory and practice of software development. In *Proc. TAPSOFT'95*, pages 3–24. LNCS 915, 1995.

[Fre72] P. Freyd. Aspects of topoi. *Bull. Austr. Math. Soc.*, (7):1–72, 1972.

[Fre73] P. Freyd. Aspects of topoi, corrections. *Bull. Austr. Math. Soc.*, (8):467–480, 1973.

[FS87] J. Fiadeiro and A. Sernadas. Structuring theories on consequence. In D. Sannella and A. Tarlecki, editors, *Recent Trend in Data Type Specification, LNCS 332*, pages 221–235. Springer Verlag, 1987.

[GB84] J. A. Goguen and R. M. Burstall. Introducing institutions. In *Proc. Logics of Programming Workshop, LNCS 164*, pages 221–256. Carnegie–Mellon, Springer, 1984.

[GB85] J. Goguen and R.M. Burstall. Institutions: Abstract model theory for computer science. Technical Report CSLI–85-30, Stanford University, 1985.

[GB92] J. Goguen and R.M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[Gog91] J. Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67, 1991.

[GTW78] J. A. Goguen, J. W. Thatcher, and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology IV: Data Structuring*, pages 80–144. Prentice Hall, 1978.

[HP89] J. M. E. Hyland and A. M. Pitts. The theory of constructions: Categorical semantics and topos-theoretic models. In *Categories in Computer Science and Logic*, pages 137–200. AMS–IMS–SIAM Joint Summer Research Conference, University of Colorado,Boulder, 1989.

[Joh77] P. T. Johnstone. *Topos Theory.* Academic Press, 1977.

[Kel74] G. M. Kelly. Review of the elements of 2–categories. *Lecture Notes in Mathematics*, (420):74–103, 1974.

[KR72] H. Kaphengst and H. Reichel. Operative Theorien und Kategorien von operativen Systemen. In *Studien zur Algebra und ihren Anwendungen*, volume 16, pages 41–56. Akademie–Verlag, 1972.

[KR77] A. Kock and G. E. Reyes. Doctrines in categorical logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 283–313. Elsevier Science Publishers B. V., North Holland, 1977.

[Law63] F. W. Lawvere. Functorial semantics of algebraic theories. In *Proc. National Academy of Science, U.S.A., 50*, pages 869–872. Columbia University, 1963.

[LS86] J. Lambek and P.J. Scott. *Introduction to higher order categorical logic*. Cambridge University Press, 1986.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, New York, 1971.

[Man76] E. Manes. *Algebraic Theories*. Springer Verlag, 1976.

[Mes89] J. Meseguer. General logics. In H.-D. Ebbinghaus et. al., editor, *Logic colloquium '87*, pages 275–329. Elsevier Science Publishers B. V.,North Holland, 1989.

[Mes90] J. Meseguer. Rewriting as a unified model of concurrency. Technical Report SRI–CSL–90-02, SRI International, Computer Science Laboratory, 1990.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *TCS*, 96:73–155, 1992.

[MG85] J. Meseguer and J. A. Goguen. Initiality, induction, and computability. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 14, pages 459–541. Cambridge University Press, 1985.

[Poi85] A. Poigné. Algebra categorically. In D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Programming*, pages 77–102. LNCS 240, Springer, 1985.

[Rei87] H. Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras*. Oxford University Press, Oxford, 1987.

[See84] R. A. G. Seely. Locally cartesian closed categories and type theory. *Mathematical Proceedings Cambridge Philosophical Society*, (95):33–48, 1984.

[See87] R. A. G. Seely. Categorical semantics for higher order polymorphic lambda calculus. *Journal of Symbolic Logic*, 52(4):969–989, December 1987.

[Tar96] A. Tarlecki. Moving between logical systems. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specifications. WADT11. Oslo Norway, September 1995*. LNCS (this volume), Springer, 1996.

[Wec92] W. Wechler. *Universal Algebra for Computer Scientists*, volume 25 of *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin, 1992.

This article was processed using the LaTeX macro package with LLNCS style