Donald Sannella and Andrzej Tarlecki

Foundations of Algebraic Specification and Formal Software Development

September 29, 2010

Springer

Contents

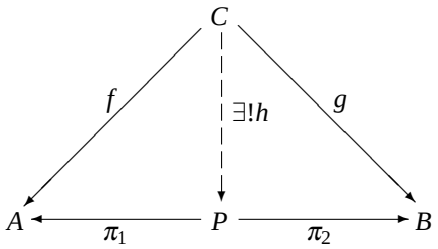
Chapter 3
Category theory

One of the main purposes of this book is to present a general, abstract theory of specifications, which is independent from the exact details of the semantic structures (algebras) used to model particular aspects of program behaviour. Appropriate mathematical tools are required to support the development of such a theory. The basics of category theory provide us with just what we need: a simple, yet powerful language that allows definitions and results to be formulated at a sufficiently general, abstract level.

The most fundamental "categorical dogma" is that for many purposes it does not really matter exactly what the objects we study are; more important are their mutual relationships. Hence, objects should never be considered on their own, they should always come equipped with an appropriate notion of a *morphism* between them. In many typical examples, the objects are sets with some additional structure imposed on them, and their morphisms are maps that preserve this structure. "Categorical dogma" states that the interesting properties of objects may be formulated purely in terms of morphisms, without referring to the internal structure of objects at all. As a very simple example, consider the following two definitions.

Definition. Given two sets A and B, the *Cartesian product* of A and B is the set $A \times B$ that consists of all the pairs of elements from A and B, respectively: $A \times B = \{\langle a, b\rangle \mid a \in A, b \in B\}$ □

Definition. Given two sets A and B, a *product* of A and B is a set P together with two functions $\pi_1 : P \to A$ and $\pi_2 : P \to B$ such that for any set C with functions $f : C \to A$ and $g : C \to B$ there exists a unique function $h : C \to P$ such that $h;\pi_1 = f$ and $h;\pi_2 = g$.



□

It is easy to see that the Cartesian product of any two sets is a product in the sense of the latter definition, where the functions π_1 and π_2 are the projections on the first and second components respectively (HINT: Define $h: C \to A \times B$ by $h(c) = \langle f(c), g(c) \rangle$ for all $c \in C$). Moreover, although a product P of two sets A and B does not have to be their Cartesian product $A \times B$ since the elements of P do not have to be pairs of objects from A and B, P is always isomorphic to $A \times B$ (there is a one-to-one correspondence between elements of P and of $A \times B$). Thus, the two definitions may be viewed as equivalent for many purposes.

The reader may feel that the former definition (of the Cartesian product) is far simpler than the latter (of a product). Indeed, to most of us, brought up to consider set-theoretic concepts as the basis of all mathematics, this is in fact the case. However, the former definition suffers from a serious deficiency: it is formulated in terms of elements and the membership relation for sets (which constitute the specific internal structure of sets). Consequently, it is very specifically oriented towards defining the Cartesian product of sets and of sets only. If we now wanted to define the Cartesian product of, say, algebras (cf. Definition 1.2.9) we would have to reformulate this definition substantially (in this case, by adding definitions of operations for product algebras). To define the Cartesian product of structures of yet another kind, yet another different version of this definition would have to be explicitly stated. It is obviously desirable to avoid such repetition of the same story for different specific kinds of objects whenever possible.

The latter definition (of a product) is quite different from this point of view. It does not make reference to the internal structure of sets at all; it defines a product of two sets entirely in terms of its relationships with these sets and with other sets. To obtain a definition of a product of two algebras, it is enough to replace "set" by "algebra" and "function" by "homomorphism". The same would apply to other kinds of structures, as long as there is an appropriate notion of a morphism between them.

The conclusion we draw from this example is that, first of all, objects of any kind should be considered together with an appropriate notion of a morphism between them, and then, that the structure imposed on the collection of objects by these morphisms should be exploited to formulate definitions at an appropriate level of generality and abstraction.

Let us have a look at another example:

Definition. A function $f: A \to B$ is *surjective* if for every $b \in B$ there exists $a \in A$ such that $b = f(a)$. □

Definition. A function $f: A \to B$ is an *epimorphism* if for any two functions $g, g': B \to C$, $f;g = f;g'$ implies $g = g'$. □

Definition. A function $f: A \to B$ is a *retraction* if there exists a function $g: B \to A$ such that $g;f = id_B$. □

All the three definitions above are equivalent: a function is surjective if and only if it is an epimorphism, if and only if it is a retraction. As with the previous example,

one may argue that the first of these definitions is very much specific to sets, and so not abstract and not general enough. The two other definitions lack this deficiency: they do not refer to the internal structure of sets, but use functions (set morphisms) to define the concept. However, the two definitions when applied to other kinds of objects (and their morphisms) may well turn out not to be equivalent. We cannot say that one of them is "right" and the other is "wrong"; they simply incorporate different aspects of what for sets is the property of "being surjective". The lesson to draw from this is that one has to be cautious when generalising a certain property to a more abstract setting. An attempt to formulate a definition at a more general level should provide us with a better understanding of the essence of the property being defined; it may well turn out, however, that there is more than one essence in it, giving several non-equivalent ways to reformulate the definition in a more abstract way.

Finding an adequate generalisation is not always easy. Sometimes even very simple notions we are accustomed to viewing as fundamental are difficult to formulate in categorical terms, as they depend in an essential way on the internal structure of the objects under consideration, which is exactly what we want to abstract from. The usual set-theoretic union operation is an example of such a notion.

Once we succeed in providing a more general version of a certain notion, it may be instantiated in many different ways. It is interesting to observe how often an adequate generalisation of an important specific concept leads to interesting instantiations in the context of objects (and morphisms between them) different from the ones we started with. Indeed, interesting instantiations in other contexts may be regarded as a test of the adequacy of the generalisation.

A more wide-ranging polemic on the advantages of category theory presented at a rather intuitive level may be found in [Gog91b].

With these remarks in mind, this chapter introduces the basic concepts and results of category theory. It is not our intention to provide a full-blown introductory text on category theory; although a few concepts are introduced which will not be used elsewhere in this book, we consciously refrain from discussing many important but more involved concepts and results. Our aim in this chapter is to provide a brief but comprehensive overview of the basics of category theory, both in order to make this book self-contained and to provide a handy reference.

3.1 Introducing categories

3.1.1 Categories

Definition 3.1.1 (Category). A *category* **K** consists of:

- a collection $|\mathbf{K}|$ of **K**-*objects*;
- for each $A, B \in |\mathbf{K}|$, a collection $\mathbf{K}(A,B)$ of **K**-*morphisms* from A to B; and

- for each $A,B,C \in |\mathbf{K}|$, a *composition operation*[1] $;:\mathbf{K}(A,B)\times\mathbf{K}(B,C)\to\mathbf{K}(A,C)$

such that:

1. for all $A,B,A',B' \in |\mathbf{K}|$, if $\langle A,B\rangle \neq \langle A',B'\rangle$ then $\mathbf{K}(A,B)\cap\mathbf{K}(A',B') = \varnothing$;
2. (*existence of identities*) for each $A \in |\mathbf{K}|$, there is a morphism $id_A \in \mathbf{K}(A,A)$ such that $id_A;g = g$ for all morphisms $g \in \mathbf{K}(A,B)$ and $f;id_A = f$ for all morphisms $f \in \mathbf{K}(B,A)$; and
3. (*associativity of composition*) for any $f \in \mathbf{K}(A,B)$, $g \in \mathbf{K}(B,C)$ and $h \in \mathbf{K}(C,D)$, $f;(g;h) = (f;g);h$. □

Notation. We refer to *objects* and *morphisms* instead of **K**-objects and **K**-morphisms when **K** is clear from the context. We write $f:A\to B$ (in **K**) for $A,B \in |\mathbf{K}|$, $f \in \mathbf{K}(A,B)$. For any $f:A\to B$, we will refer to A as the *source* or *domain*, and to B as the *target* or *codomain* of f. The collection of all morphisms of **K** will be (ambigously) denoted by **K** as well, i.e., $\mathbf{K} = \bigcup_{A,B\in|\mathbf{K}|}\mathbf{K}(A,B)$. □

The above is just one of several possible equivalent definitions of a category. For example, the identities, the existence of which is required in (2), are sometimes considered as part of the structure of a category.

Exercise 3.1.2. Prove that in any category, identities are unique. □

The notion of a category is very general. Accepting the categorical dogma that objects of any kind come equipped with a notion of morphism between them, it is difficult to think of a collection of objects and accompanying morphisms that do not form a category. Almost always there is a natural operation of morphism composition, which obeys two of the basic requirements above: it has identities and is associative. Perhaps requirement (1), which allows us to unambigously identify the source and target of any morphism, is the most technical and hence least intuitively appealing. But even in cases where the same entity may be viewed as a morphism between different objects, this entity can always be equipped with an explicit indication of the source and target of the morphism (cf. Example 3.1.6), thus satisfying requirement (1).

In the rest of this subsection we give a number of examples of categories. We start with some rather trivial examples, mainly of formal interest, and only then define some more typically considered categories. Further examples, which are often more complex, may be found in the following sections of this chapter (and in later chapters, see e.g. Section 10.3 for somewhat more complex examples).

Example 3.1.3 (Preorder categories). A binary relation $\leq \subseteq X\times X$ is a *preorder on* X if:

- $x \leq x$ for all $x \in X$; and
- $x \leq y \wedge y \leq z \Rightarrow x \leq z$ for all $x,y,z \in X$.

[1] We will use semicolon ; to denote composition of morphisms in any category, just as we used it for composition of functions and homomorphisms in the preceding chapters. Composition will always be written in diagrammatic order: $f;g$ is to be read as "f followed by g".

A *preorder* category is a category that has at most one morphism with any given source and target.

Every preorder $\leq \subseteq X \times X$ gives rise to a preorder category $\mathbf{K}_{\leq}$ where $|\mathbf{K}_{\leq}| = X$ and $\mathbf{K}_{\leq}(x,y)$ has exactly one element if $x \leq y$ and is empty otherwise.

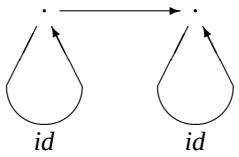
This definition does not identify the category $\mathbf{K}_{\leq}$ unambigously, since different elements may be used as morphisms in $\mathbf{K}_{\leq}(x,y)$ for $x \leq y$. However, we will not worry here about the exact nature of morphisms (nor objects) in a category, and we will treat this and similar definitions below as sufficient. More formally, all categories satisfying the above requirements are isomorphic in the technical sense to be discussed in Section 3.4 (cf. Definition 3.4.68).

Here are some trivial examples of preorder categories:

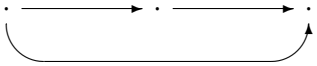
0: (the empty category)

1:

2:

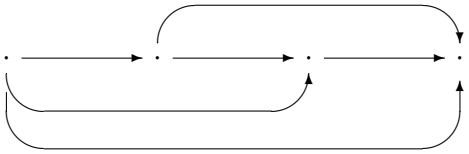


3:



(+ identities)

4:



(+ identities)

⋮ ⋮

Exercise. How many morphisms does **n** have? □

Example 3.1.4 (Discrete category). A category **K** is *discrete* whenever for all $A, B \in |\mathbf{K}|$, $\mathbf{K}(A,B)$ is empty if $A \neq B$ and contains exactly one element (the identity) otherwise.

Any collection of objects X gives rise to a discrete category $\mathbf{K}_X$ where $|\mathbf{K}_X| = X$. □

Example 3.1.5 (Monoid category). A category **K** is a *monoid* if **K** has exactly one object.

A set X together with a function $; : X \times X \to X$ and a distinguished element $id \in X$ is a *monoid* $\langle X, ;, id \rangle$ if $(x;y);z = x;(y;z)$ and $id;x = x;id = x$ for all $x, y, z \in X$. Every monoid $\langle X, ;, id \rangle$ gives rise to a monoid (category) having morphisms X and composition $;$. □

Example 3.1.6 (Set, the category of sets). The category **Set** of sets with functions as morphisms is defined as follows:

Objects of **Set**: sets;
Morphisms of **Set**: functions; however, to ensure that the requirements stated in Definition 3.1.1 are satisfied (disregarding the particular mathematical representation of the concept of a function one uses), we will always consider functions with explicitly given domain and codomain. Thus, a morphism in the category **Set** with source A and target B is a triple $\langle A, f, B \rangle$, where $f : A \to B$ is a function. □

Example 3.1.7 ($\mathbf{Set}^S$, the category of S-sorted sets). For any set S, the category $\mathbf{Set}^S$ of S-sorted sets is defined as follows:

Objects of $\mathbf{Set}^S$: S-sorted sets;
Morphisms of $\mathbf{Set}^S$: S-sorted functions (with explicitly given domain and codomain). □

Example 3.1.8 ($\mathbf{Alg}(\Sigma)$, the category of Σ-algebras). For any signature Σ, the category $\mathbf{Alg}(\Sigma)$ of Σ-algebras is defined as follows:

Objects of $\mathbf{Alg}(\Sigma)$: Σ-algebras;
Morphisms of $\mathbf{Alg}(\Sigma)$: Σ-homomorphisms (with explicitly given domain and codomain). □

Example 3.1.9 (CPO, the category of complete partial orders). The category **CPO** of complete partial orders[2] and continuous functions between them is defined as follows:

Objects of **CPO**: complete partial orders, i.e., partially ordered sets $\langle X, \leq \rangle$ such that any countable chain $x_0 \leq x_1 \leq \ldots$ in $\langle X, \leq \rangle$ has a least upper bound $\bigsqcup_{i \geq 0} x_i$;
Morphisms of **CPO**: continuous functions, i.e., functions that preserve least upper bounds of countable chains. □

Exercise 3.1.10. Complete the above examples by formalising composition in the obvious way. Indicate identities and prove associativity of composition. □

Example 3.1.11 (AlgSig, the category of algebraic signatures). The category **AlgSig** of (algebraic) signatures is defined as follows:

[2] Cpos and continuous functions as defined here are often referred to as ω-cpos and ω-continuous functions, respectively.

Objects of **AlgSig**: signatures;
Morphisms of **AlgSig**: signature morphisms;
Composition in **AlgSig**: for any $\sigma: \Sigma \to \Sigma'$ and $\sigma': \Sigma' \to \Sigma''$, their composition $\sigma;\sigma': \Sigma \to \Sigma''$ is given by $(\sigma;\sigma')_{sorts} = \sigma_{sorts};\sigma'_{sorts}$ and $(\sigma;\sigma')_{ops} = \sigma_{ops};\sigma'_{ops}$, cf. Exercise 1.5.3. □

Exercise 3.1.12 ($\mathbf{AlgSig}^{der}$, the category of signatures with derived morphisms). Recall the concept of a derived signature morphism from Definition 1.5.14. Define the category $\mathbf{AlgSig}^{der}$ of algebraic signatures with derived signature morphisms. Use Exercise 1.5.18 to define composition of derived signature morphisms. □

Example 3.1.13 ($\mathbf{T}_\Sigma$, the category of substitutions over a signature Σ). Recall (cf. Section 1.4) that for any signature $\Sigma = \langle S, \Omega \rangle$ and S-sorted set of variables X, $T_\Sigma(X)$ is the algebra of terms over Σ with variables X. $T_\Sigma(X)$ is characterised up to isomorphism by the property that for any Σ-algebra A, any S-sorted map $v: X \to |A|$ uniquely extends to a Σ-homomorphism $v^\#: T_\Sigma(X) \to A$ (Facts 1.4.4 and 1.4.10).

For any algebraic signature Σ, the category $\mathbf{T}_\Sigma$ of substitutions over Σ is defined as follows (cf. Exercise 1.4.9):

Objects of $\mathbf{T}_\Sigma$: S-sorted sets (of variables);
Morphisms of $\mathbf{T}_\Sigma$: for any sets X and Y, a morphism θ from X to Y is a substitution of terms with variables Y for variables X, i.e., an S-sorted function $\theta: X \to |T_\Sigma(Y)|$;
Composition in $\mathbf{T}_\Sigma$: given any sets X, Y and Z, and morphisms $\theta: X \to Y$ and $\theta': Y \to Z$ in $\mathbf{T}_\Sigma$, i.e., functions $\theta: X \to |T_\Sigma(Y)|$ and $\theta': Y \to |T_\Sigma(Z)|$, their composition $\theta;\theta': X \to Z$ is the function $\theta;\theta': X \to |T_\Sigma(Z)|$ defined by $(\theta;\theta')_s(x) = (\theta')^\#_s(\theta_s(x))$ for all $s \in S$, $x \in X_s$. □

Exercise 3.1.14 ($\mathbf{T}_\Sigma/\Phi$, the category of substitutions over Σ modulo equations Φ). Generalise the above definition of the category of substitutions by considering terms up to an equivalence generated by a set of equations. That is, for any algebraic signature $\Sigma = \langle S, \Omega \rangle$ and set Φ of Σ-equations, for any S-sorted set of variables X define two terms $t_1, t_2 \in |T_\Sigma(X)|_s$ (for any sort $s \in S$) to be equivalent, written $t_1 \equiv t_2$, if $\Phi \vdash_\Sigma \forall X \bullet t_1 = t_2$ (cf. Section 2.4). Now, by analogy with the category of substitutions, define the category $\mathbf{T}_\Sigma/\Phi$ to have S-sorted sets as objects and substitutions modulo Φ as morphisms. A substitution of terms modulo Φ with variables Y for variables X is an S-sorted function $\theta: X \to (|T_\Sigma(Y)|/\equiv)$. Composition in $\mathbf{T}_\Sigma/\Phi$ is defined analogously as in $\mathbf{T}_\Sigma$, by choosing a representative of each of the equivalence classes assigned to variables: given $\theta: X \to (|T_\Sigma(Y)|/\equiv)$ and $\theta': Y \to (|T_\Sigma(Z)|/\equiv)$, $\theta;\theta': X \to (|T_\Sigma(Z)|/\equiv)$ maps any $x \in X$ to $(\theta')^\#(t)$, where $\theta(x) = [t]_\equiv$ (show that the result does not depend on the choice of the representative $t \in \theta(x)$). □

Exercise 3.1.15 ($\mathbf{T}_{\Sigma,\Phi}$, the algebraic $\langle \Sigma, \Phi \rangle$-theory). Building on the definition of the category of substitutions modulo a set of equations sketched above, abstract away from the actual names of variables used in the objects of $\mathbf{T}_\Sigma/\Phi$ by listing them in some particular order, as in derived signatures (cf. Definition 1.5.13). That is, for

any algebraic signature $\Sigma = \langle S, \Omega \rangle$ and set Φ of Σ-equations, define the category $\mathbf{T}_{\Sigma,\Phi}$ with sequences $s_1 \dots s_n \in S^*$ of sort names as objects. A morphism in $\mathbf{T}_{\Sigma,\Phi}$ from $s_1 \dots s_n \in S^*$ to $s'_1 \dots s'_m \in S^*$ is an n-tuple $\langle [t_1]_{\equiv}, \dots, [t_n]_{\equiv} \rangle$ of terms modulo Φ, where the equivalence $\equiv$ is sketched in Exercise 3.1.14 above, and for $i = 1, \dots, n$, $t_i \in |T_\Sigma(I_{s'_1 \dots s'_m})|_{s_i}$, with $I_{s'_1 \dots s'_m} = \{\boxed{1}{:}s'_1, \dots, \boxed{m}{:}s'_m\}$. The composition in $\mathbf{T}_{\Sigma,\Phi}$ is given by substitution on representatives of equivalence classes (the position of a term in a tuple identifies the variable it is to be substituted for). $\mathbf{T}_{\Sigma,\Phi}$ is usually referred to as the *algebraic theory* over Σ generated by Φ.[3] □

3.1.1.1 Foundations

In the above, and in the definition of a category in particular, we have very cautiously used the non-technical term *collection*, and talked of *collections* of objects and morphisms. This allowed us to gloss over the issue of the choice of appropriate set-theoretical foundations for category theory. Even a brief look at the examples above indicates that we could not have been talking here just of *sets* (in the sense of Zermelo-Fraenkel set theory): we want to consider categories like **Set**, where the collection of objects consists of all sets, and so cannot be a set itself. Using *classes* (collections of sets that are possibly too "large" to be sets themselves, as in Bernays-Gödel set theory) might seem more promising, since if we replace the term "collection" by "class" in Definition 3.1.1 then at least examples of categories like **Set** would be covered. However, this is not enough either, since even in this simple presentation of the basics of category theory we will encounter some categories (like **Cat**, the category of "all" categories, and functor categories defined later in this chapter) where objects themselves are proper classes and the collection of objects forms a "conglomerate" (a collection of classes that is too "large" to be a class, cf. [HS73]). We refer to [Bén85] for a careful analysis of the basic requirements imposed on a set theory underlying category theory.

Perhaps the most traditional solution to the problem of set-theoretic foundations for category theory is sketched in [Mac71]. The idea is to work within a hierarchy of *set universes* $\langle \mathbf{U}_n \rangle_{n \geq 0}$, where each universe $\mathbf{U}_n$, $n \geq 0$, is closed under the standard set-theoretic operations, and is an element of the next universe in the hierarchy, $\mathbf{U}_n \in \mathbf{U}_{n+1}$. Then there is a notion of category corresponding to each level of the hierarchy, and one is required to indicate at which level of the hierarchy one is working at any given moment.

However, in our view such pedantry would hide the intuitive appeal of "naive" category theory. We will therefore ignore the issue of set-theoretic foundations for category theory in the sequel, with just one exception: we define what it means for a category to be (locally) small and use this to occasionally warn the reader about potential foundational hazards.

[3] In the literature, the algebraic theory over Σ generated by Φ is often defined with substitutions considered as morphisms in the opposite direction, i.e., as the category $\mathbf{T}^{op}_{\Sigma,\Phi}$ opposite to $\mathbf{T}_{\Sigma,\Phi}$ (cf. Definition 3.1.21 below).

Definition 3.1.16 (Small category). A category $\mathbf{K}$ is *locally small* if for any $A, B \in |\mathbf{K}|$, $\mathbf{K}(A,B)$ is a set (an element of the lowest-level universe $\mathbf{U}_0$); $\mathbf{K}$ is *small* if in addition $|\mathbf{K}|$ is a set as well. □

3.1.2 Constructing categories

In the examples of the previous subsection, each category was constructed "from scratch" by explicitly defining its objects and morphisms and their composition. Category theory also provides numerous ways of modifying a given category to yield a different one, and of putting together two or more categories to obtain a more complicated one. Some of the simplest examples are given in this subsection.

3.1.2.1 Subcategories

Definition 3.1.17 (Subcategory). A category **K1** is a *subcategory* of a category **K2** if $|\mathbf{K1}| \subseteq |\mathbf{K2}|$ and $\mathbf{K1}(A,B) \subseteq \mathbf{K2}(A,B)$ for all objects $A, B \in |\mathbf{K1}|$, with composition and identities in **K1** the same as in **K2**. **K1** is a *full* subcategory of **K2** if additionally $\mathbf{K1}(A,B) = \mathbf{K2}(A,B)$ for all $A, B \in |\mathbf{K1}|$. **K1** is a *wide* subcategory of **K2** if $|\mathbf{K1}| = |\mathbf{K2}|$. □

For any category $\mathbf{K}$, any collection $X \subseteq |\mathbf{K}|$ of objects of $\mathbf{K}$ determines a full subcategory $\mathbf{K}|_X$ of $\mathbf{K}$, defined by $|\mathbf{K}|_X| = X$. Whenever convenient, if $\mathbf{K}$ is evident from the context, we will identify collections $X \subseteq |\mathbf{K}|$ with $\mathbf{K}|_X$.

Example 3.1.18 (FinSet, the category of finite sets). The category **FinSet** of finite sets is defined as follows:

Objects of **FinSet**: finite sets;
Morphisms and composition in **FinSet**: as in **Set**.

FinSet is a full subcategory of **Set**. □

Example 3.1.19. The category of single-sorted signatures is a full subcategory of the category **AlgSig** of (many-sorted) signatures.

The discrete category of sets is a subcategory of the category of sets with inclusions as morphisms, which is a subcategory of the category of sets with injective functions as morphisms, which is a subcategory of **Set**.

For any signature Σ and set Φ of Σ-equations, the class $Mod_\Sigma(\Phi)$ of Σ-algebras that satisfy Φ determines a full subcategory of $\mathbf{Alg}(\Sigma)$, which we denote by $\mathbf{Mod}(\Sigma, \Phi)$. □

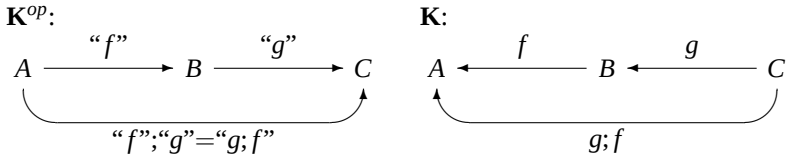
Exercise 3.1.20. Give an example of two categories **K1**, **K2** such that $|\mathbf{K1}| \subseteq |\mathbf{K2}|$, $\mathbf{K1}(A,B) \subseteq \mathbf{K2}(A,B)$ for all objects $A, B \in |\mathbf{K1}|$, with composition in **K1** the same as in **K2**, but such that **K1** is *not* a subcategory of **K2**. □

3.1.2.2 Opposite categories and duality

One of the fundamental theorems of lattice theory (cf. e.g. [DP90]) is the so-called *duality principle*. Any statement in the language of lattice theory has a dual, obtained by systematically replacing greatest lower bounds by least upper bounds and vice versa. The duality principle states that the dual of any theorem of lattice theory is a theorem as well. In a sense, this allows the number of proofs in lattice theory to be cut by half: proving a fact gives its dual "for free". A very similar phenomenon occurs in category theory; in fact, the duality principle of lattice theory may be viewed as a consequence of a more general duality principle of category theory. Replacing greatest lower bounds by least upper bounds and vice versa is generalised here to the process of "reversing morphisms".

Definition 3.1.21 (Opposite category). The *opposite category* of a category $\mathbf{K}$ is the category $\mathbf{K}^{op}$ where:

Objects of $\mathbf{K}^{op}$: $|\mathbf{K}^{op}| = |\mathbf{K}|$;
Morphisms of $\mathbf{K}^{op}$: $\mathbf{K}^{op}(A,B) = \mathbf{K}(B,A)$ for all $A,B \in |\mathbf{K}^{op}|$;
Composition in $\mathbf{K}^{op}$: for $f \in \mathbf{K}^{op}(A,B)$ (i.e., $f \in \mathbf{K}(B,A)$) and $g \in \mathbf{K}^{op}(B,C)$ (i.e., $g \in \mathbf{K}(C,B)$), $f;g \in \mathbf{K}^{op}(A,C)$ is $g;f \in \mathbf{K}(C,A)$.



□

Exercise 3.1.22. Check that:

1. $\mathbf{K}^{op}$ is a category.
2. $(\mathbf{K}^{op})^{op} = \mathbf{K}$.
3. Identities in $\mathbf{K}^{op}$ are the same as in $\mathbf{K}$. □

If W is a categorical concept (property, statement, ...) then its *dual*, *co-W*, is obtained by reversing all the morphisms in W. This idea may be formalised in two ways. The first is to introduce a formal language of category theory, and then define the operation of forming a dual as an operation on formal statements in this language. The other is to formally interpret *co-W* in a category $\mathbf{K}$ as W in the category $\mathbf{K}^{op}$. Since formalising the language of category theory is beyond the scope of this book (but cf. [Mac71] or [Hat82]), we take the second option here and will rely on an intuitive understanding of duality in the sequel. For example, consider the following property of objects in a category:

$$P(X): \text{ for any object } Y \text{ there is a morphism } f: Y \to X.$$

Then:

$$\textit{co-}P(X): \text{ for any object } Y \text{ there is a morphism } f: X \to Y.$$

Note that indeed *co-*$P(X)$ in any category $\mathbf{K}$ amounts to $P(X)$ in $\mathbf{K}^{op}$.

Since any category is the opposite of a certain category (namely, of its opposite), the following fact holds:

Fact 3.1.23 (Duality principle). *If W holds for all categories then* co-*W holds for all categories as well.* □

3.1.2.3 Product categories

Definition 3.1.24 (Product category). For any two categories **K1** and **K2**, the *product category* $\mathbf{K1} \times \mathbf{K2}$ is defined by:

Objects of $\mathbf{K1} \times \mathbf{K2}$: $|\mathbf{K1} \times \mathbf{K2}| = |\mathbf{K1}| \times |\mathbf{K2}|$ (the Cartesian product);
Morphisms of $\mathbf{K1} \times \mathbf{K2}$: for all $A, A' \in |\mathbf{K1}|$ and $B, B' \in |\mathbf{K2}|$,
$\mathbf{K1} \times \mathbf{K2}(\langle A, B\rangle, \langle A', B'\rangle) = \mathbf{K1}(A, A') \times \mathbf{K2}(B, B')$;
Composition in $\mathbf{K1} \times \mathbf{K2}$: for $f : A \to A'$ and $f' : A' \to A''$ in **K1**, $g : B \to B'$ and $g' : B' \to B''$ in **K2**, $\langle f, g\rangle ; \langle f', g'\rangle = \langle f ; f', g ; g'\rangle$. □

Exercise 3.1.25. Identify the category to which each semicolon in the above definition of composition in $\mathbf{K1} \times \mathbf{K2}$ refers. Then show that $\mathbf{K1} \times \mathbf{K2}$ is indeed a category. □

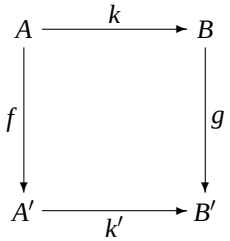
Exercise 3.1.26. Define $\mathbf{K}^n$, where $\mathbf{K}$ is a category and $n \geq 1$. What would you suggest for $n = 0$? □

3.1.2.4 Morphism categories

Definition 3.1.27 (Morphism category). For any category $\mathbf{K}$, the *category* $\mathbf{K}^{\to}$ *of* $\mathbf{K}$*-morphisms* is defined by:

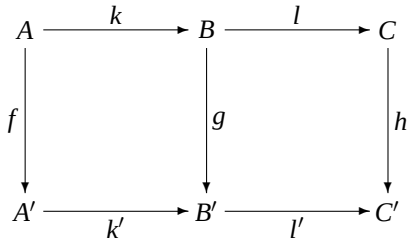
Objects of $\mathbf{K}^{\to}$: $\mathbf{K}$-morphisms;
Morphisms of $\mathbf{K}^{\to}$: a morphism in $\mathbf{K}^{\to}$ from $f : A \to A'$ (in $\mathbf{K}$) to $g : B \to B'$ (in $\mathbf{K}$) is a pair $\langle k, k'\rangle$ of $\mathbf{K}$-morphisms where $k : A \to B$ and $k' : A' \to B'$ such that $k ; g = f ; k'$;
Composition in $\mathbf{K}^{\to}$: $\langle k, k'\rangle ; \langle l, l'\rangle = \langle k ; l, k' ; l'\rangle$. □

The requirement in the definition of a morphism in $\mathbf{K}^{\to}$ may be more illustratively restated as the requirement that the following diagram commutes in the category $\mathbf{K}$:



For now, we will rely on an intuitive understanding of the concept of a diagram in a category; see Section 3.2.5 for a formal definition. We say that a diagram in a category *commutes* (or, *is commutative*) if for any two paths with the same source and target nodes, the composition of morphisms along each of the two paths yields the same result.

Drawing diagrams and *chasing* a diagram in order to prove that it is commutative is one of the standard and intuitively most appealing techniques used in category theory. For example, to justify Definition 3.1.27 above it is essential to show that the composition of two morphisms in $\mathbf{K}^{\rightarrow}$ as defined there yields a morphism in $\mathbf{K}^{\rightarrow}$. This may be done by *pasting together* two diagrams like the one above along a common edge, obtaining the following diagram:

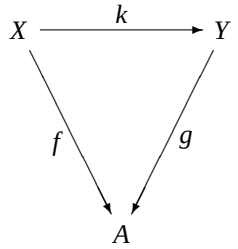


A simple argument may now be used to show that if the two simpler diagrams are commutative then the above diagram obtained by pasting them together along the edge labelled by g commutes as well:

$$f;(k';l') = (f;k');l' = (k;g);l' = k;(g;l') = k;(l;h) = (k;l);h$$

Definition 3.1.28 (Slice category). Let $\mathbf{K}$ be a category with $A \in |\mathbf{K}|$. The *category* $\mathbf{K}{\downarrow}A$ *of* $\mathbf{K}$*-objects over* A (or, the *slice of* $\mathbf{K}$ *over* A) is defined by:

Objects of $\mathbf{K}{\downarrow}A$: pairs $\langle X, f\rangle$ where $X \in |\mathbf{K}|$ and $f \in \mathbf{K}(X,A)$;
Morphisms of $\mathbf{K}{\downarrow}A$: a morphism from $\langle X, f\rangle$ to $\langle Y, g\rangle$ is a $\mathbf{K}$-morphism $k: X \to Y$ such that $k;g = f$:



Composition in $\mathbf{K}{\downarrow}A$: as in $\mathbf{K}$. □

Exercise 3.1.29. Show that $\mathbf{K}{\downarrow}A$ may be constructed as a subcategory of $\mathbf{K}^{\rightarrow}$. Is it full? □

Exercise 3.1.30. Define $\mathbf{K}{\uparrow}A$, the category of $\mathbf{K}$-objects *under* A. Compare $(\mathbf{K}{\downarrow}A)^{op}$, $\mathbf{K}^{op}{\downarrow}A$ and $(\mathbf{K}^{op}{\downarrow}A)^{op}$ with $\mathbf{K}{\uparrow}A$. □

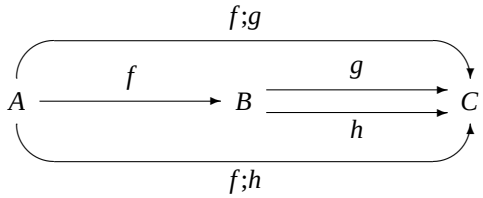
3.1.3 Category-theoretic definitions

In this section we will give a few simple examples of how certain special morphisms may be characterised in a style that is typical for category-theoretic definitions. As indicated in the introduction to this chapter, the idea is to abstract away from the "internal" properties of objects and morphisms, characterising them entirely in categorical language by referring only to arbitrary objects and morphisms of the category under consideration. Such definitions may be formulated for an arbitrary category, and then instantiated to a particular one when necessary. We will also indicate a few basic properties of the concepts we introduce that hold in any category.

Throughout this section, let **K** be an arbitrary but fixed category. Morphisms and objects we refer to below are those of **K**, unless explicitly qualified otherwise.

3.1.3.1 Epimorphisms and monomorphisms

Definition 3.1.31 (Epimorphism). A morphism $f: A \to B$ is an *epimorphism* (or *is epi*) if for all $g: B \to C$ and $h: B \to C$, $f;g = f;h$ implies $g = h$.



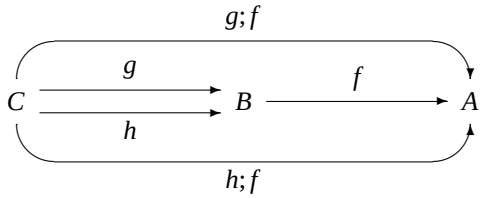
□

Example 3.1.32. In **Set**, f is epi iff f is surjective. □

There are "natural" categories in which epimorphisms need not be surjective. For example:

Exercise 3.1.33. Recall the category **CPO** of complete partial orders and continuous functions introduced in Example 3.1.9. Give an example of a continuous function that is an epimorphism in **CPO** even though it is not surjective. Try to characterise epimorphisms in this category. □

Definition 3.1.34 (Monomorphism). A morphism $f: B \to A$ is a *monomorphism* (or *is mono*) if for all $g: C \to B$ and $h: C \to B$, $g;f = h;f$ implies $g = h$.



□

Example 3.1.35. In **Set**, f is mono iff f is injective. □

Note that mono means the same as co-epi, i.e., f is mono in $\mathbf{K}$ iff f is epi in $\mathbf{K}^{op}$.

Fact 3.1.36.

1. *If $f:A \to B$ and $g:B \to C$ are mono then $f;g:A \to C$ is mono.*
2. *For any $f:A \to B$ and $g:B \to C$, if $f;g:A \to C$ is mono then f is mono.*

Proof. The proof is rather straightforward, and significantly more complex proofs will be omitted in the rest of this chapter. We present it here explicitly only as a simple example of the style of argument, very common in category-theoretic proofs, exploiting the most basic properties of composition in an arbitrary category.

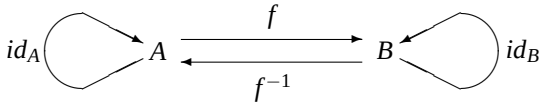
1. According to Definition 3.1.34, we have to show that for any $h,h':D \to A$ if $h;(f;g) = h';(f;g)$ then $h = h'$. So, suppose $h;(f;g) = h';(f;g)$. Then, since composition is associative, $(h;f);g = (h';f);g$. Consequently, since g is mono, by Definition 3.1.34, $h;f = h';f$. Thus, using the fact that f is mono, we can indeed derive $h = h'$.
2. Similarly as in the previous case: suppose that for some $h,h':D \to A$, $h;f = h';f$. Then also $(h;f);g = (h';f);g$, and so $h;(f;g) = h';(f;g)$. Now, since $f;g$ is mono, it follows directly from the definition that indeed $h = h'$.

□

Exercise 3.1.37. Dualise both parts of Fact 3.1.36. Formulate the dual proofs and check that they are indeed sound. □

3.1.3.2 Isomorphic objects

Definition 3.1.38 (Isomorphism). A morphism $f:A \to B$ is an *isomorphism* (or *iso*) if there is a morphism $f^{-1}:B \to A$ such that $f;f^{-1} = id_A$ and $f^{-1};f = id_B$. The morphism $f^{-1}:B \to A$ is called the *inverse* of f, and the objects A and B are called *isomorphic*. We write $f:A \cong B$ or just $A \cong B$.



□

Exercise 3.1.39. Show that the inverse of a morphism, if it exists, is unique. □

Note that iso means the same as co-iso, that is, isomorphism is a *self-dual* concept.

Exercise 3.1.40. Check that if $f:A \to B$ and $g:B \to C$ are iso then $f;g:A \to C$ is iso as well. □

In **Set**, a morphism is iso iff it is both epi and mono. However, this property does not carry over to an arbitrary category:

Exercise 3.1.41. Show that if f is iso then f is both epi and mono. The converse is not true in general; give a counterexample. □

Exercise 3.1.42. We say that a morphism $f:A \to B$ is a *retraction* if there is a morphism $g:B \to A$ such that $g;f = id_B$. Dually, a morphism $f:A \to B$ is a *coretraction* if there is a morphism $g:B \to A$ such that $f;g = id_A$. Show that:

1. A morphism is iso iff it is both a retraction and a coretraction.
2. Every retraction is epi.
3. A morphism is iso iff it is an epi coretraction.

Dualise the above facts. □

It is easy to see that any two isomorphic objects have the same "categorical properties". Intuitively, such objects have abstractly the same structure and so are indistinguishable within the given category (which does not mean that isomorphic objects cannot have different "non-categorical" properties, cf. Example 1.3.12). Indeed, an isomorphism and its inverse determine one-to-one mappings between morphisms going into and coming out of isomorphic objects. Hence, categorical definitions of objects define them only "up to isomorphism". The following section provides typical examples of this phenomenon.

3.2 Limits and colimits

In this section we show how certain special objects in an arbitrary category together with their "characteristic" morphisms may be defined in purely categorical terms by so-called *universal properties*; we hope that the reader will recognise the pattern in the example definitions below. Sections 3.2.1–3.2.4 present some typical instances of this, introducing the most commonly used cases of the general *limit construction* and its dual, which are then presented in their full generality in Section 3.2.5. In most of the cases in this section we will explicitly spell out the duals of the concepts introduced, since many of them have interesting instances in some common categories (and are traditionally given independent names).

Throughout this section, let **K** be an arbitrary but fixed category. Morphisms and objects we refer to are those of **K**, unless explicitly qualified otherwise.

3.2.1 Initial and terminal objects

Definition 3.2.1 (Initial object). An object $I \in |\mathbf{K}|$ is *initial in* **K** if for each $A \in |\mathbf{K}|$ there is exactly one morphism from I to A. □

Example 3.2.2. The empty set $\varnothing$ is initial in **Set**. The algebra T_Σ of ground Σ-terms is initial in $\mathbf{Alg}(\Sigma)$, for any signature $\Sigma \in |\mathbf{AlgSig}|$.

Recall the definition of an initial model of an equational specification (Definition 2.5.13). For any signature Σ and a set Φ of Σ-equations, the initial model of $\langle \Sigma, \Phi \rangle$ (which exists by Theorem 2.5.14) is an initial object in the category $\mathbf{Mod}(\Sigma, \Phi)$ (as defined in Example 3.1.19). □

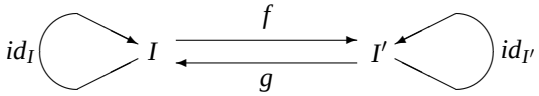
Exercise 3.2.3. What is an initial object in **AlgSig**? Look for initial objects in other categories. □

Fact 3.2.4.

1. *Any two initial objects in* $\mathbf{K}$ *are isomorphic.*
2. *If* I *is initial in* $\mathbf{K}$ *and* I' *is isomorphic to* I *then* I' *is initial in* $\mathbf{K}$ *as well.*

Proof. The proof is rather straightforward. We present it here explicitly only as a simple example of the style of argument, very common in category-theoretic proofs, which exploits universality (a special case of which is the property used in the definition of an initial object). The requirement that there *exists* a morphism satisfying a certain property is used to construct some diagrams, and then the *uniqueness* of this morphism is used to show that the diagrams constructed commute.

1. Suppose that $I, I' \in |\mathbf{K}|$ are two initial objects in $\mathbf{K}$. Then, by the initiality of I, there exists a morphism $f : I \to I'$. Similarly, by the initiality of I', there exists a morphism $g : I' \to I$. Thus, we have constructed the following diagram:



   Now, by the initiality of I, there is a *unique* morphism from I to I, and so $id_I = f;g$. Similarly, $id_{I'} = g;f$. Thus f is an isomorphism (with inverse g) and I and I' are indeed isomorphic.
2. Suppose that $I \in |\mathbf{K}|$ is initial in $\mathbf{K}$, and let $i : I \to I'$ be an isomorphism with inverse $i^{-1} : I' \to I$. Consider an arbitrary object $A \in |\mathbf{K}|$. By the "existence part" of the initiality property of I, we know that there exists a morphism $f : I \to A$. Hence, there exists a morphism from I' to A as well, namely $i^{-1};f : I' \to A$. Then, let $f' : I' \to A$ be an arbitrary morphism from I' to A. By the "uniqueness part" of the initiality property of I, $f = i;f'$, and so $i^{-1};f = i^{-1};(i;f') = (i^{-1};i);f' = id_{I'};f' = f'$. This shows that $i^{-1};f$ is the only morphism from I' to A, and so that I' is indeed initial in $\mathbf{K}$.

□

The last fact indicates that the initiality property identifies an object up to isomorphism. As argued in Section 3.1.3.2, in category theory this is the most exact characterisation of an object we may expect. In the following we will speak of "the" initial object meaning an initial object identified up to isomorphism. We adopt the same convention in the many similar cases introduced in the sequel.

3.2.1.1 Dually:

Definition 3.2.5 (Terminal object). An object $1 \in |\mathbf{K}|$ is *terminal in* $\mathbf{K}$ if for each $A \in |\mathbf{K}|$ there is exactly one morphism from A to 1. □

Note that terminal means the same as co-initial.

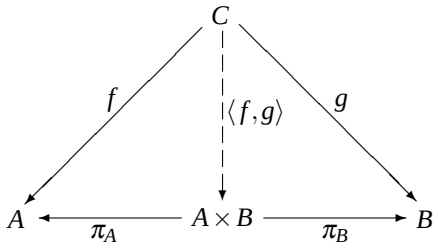
Exercise 3.2.6. Are there any terminal objects in **Set**, $\mathbf{Alg}(\Sigma)$ or **AlgSig**? What about terminal objects in $\mathbf{AlgSig}^{der}$?

Recall the definition of a terminal (final) model of an equational specification (Definition 2.7.12). Restate it using the notion of a terminal object as defined above. □

Exercise 3.2.7. Dualise Fact 3.2.4. □

3.2.2 Products and coproducts

Definition 3.2.8 (Product). A *product* of two objects $A, B \in |\mathbf{K}|$ is an object $A \times B \in |\mathbf{K}|$ together with a pair of morphisms $\pi_A : A \times B \to A$ and $\pi_B : A \times B \to B$ such that for any object $C \in |\mathbf{K}|$ and pair of morphisms $f : C \to A$ and $g : C \to B$ there is exactly one morphism $\langle f, g \rangle : C \to A \times B$ such that the following diagram commutes:



□

Example 3.2.9. In **Set**, the Cartesian product of A and B is a product $A \times B$, where π_A, π_B are the projection functions. For any signature Σ, products in $\mathbf{Alg}(\Sigma)$ are defined analogously (cf. Definition 1.2.9). □

Exercise 3.2.10. What is the product of two objects in a preorder category? □

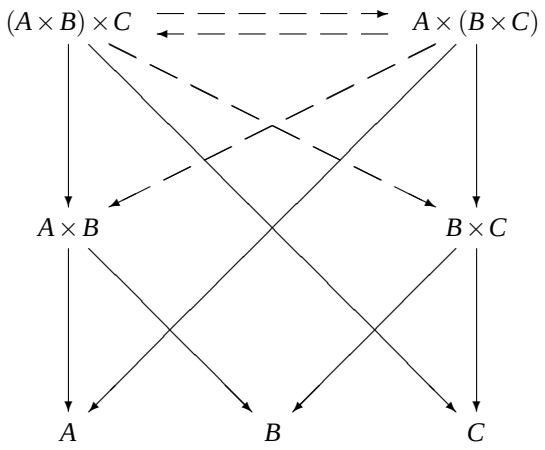
Exercise 3.2.11. Show that any two products of $A, B \in |\mathbf{K}|$ are isomorphic. □

Exercise 3.2.12. Suppose that $A, B \in |\mathbf{K}|$ have a product. Given $f : C \to A$ and $g : C \to B$, and hence $\langle f, g \rangle : C \to A \times B$, show that for any $h : D \to C$, $h ; \langle f, g \rangle = \langle h ; f, h ; g \rangle$. □

Exercise 3.2.13. Prove that:

1. $A \times B \cong B \times A$ for any $A, B \in |\mathbf{K}|$.

2. $(A \times B) \times C \cong A \times (B \times C)$ for any $A, B, C \in |\mathbf{K}|$. HINT: The following diagram might be helpful:

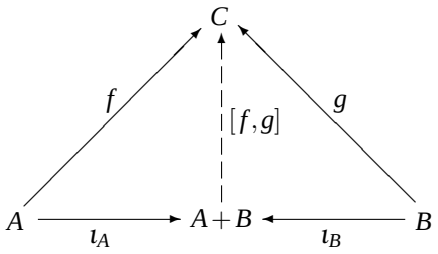


□

Exercise 3.2.14. Define the product of an arbitrary family of **K**-objects. What is the product of the empty family? □

3.2.2.1 Dually:

Definition 3.2.15 (Coproduct). A *coproduct* of two objects $A, B \in |\mathbf{K}|$ is an object $A+B \in |\mathbf{K}|$ together with a pair of morphisms $\iota_A : A \to A+B$ and $\iota_B : B \to A+B$ such that for any object $C \in |\mathbf{K}|$ and pair of morphisms $f : A \to C$ and $g : B \to C$ there is exactly one morphism $[f,g] : A+B \to C$ such that the following diagram commutes:



□

Example 3.2.16. In **Set**, the disjoint union of sets A and B is their coproduct $A+B$, where ι_A, ι_B are the injections. Similarly, in **AlgSig**, the (componentwise) disjoint union of algebraic signatures Σ and Σ' is their coproduct $\Sigma + \Sigma'$, where ι_A, ι_B are the obvious injections. □

Note that coproduct means the same as co-product.

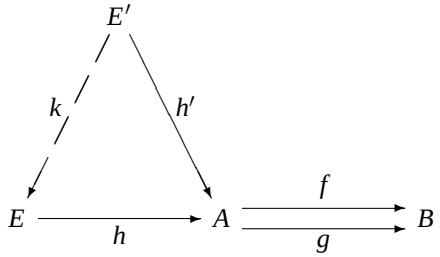
Exercise 3.2.17. Dualise the exercises for products. □

Exercise 3.2.18. For any algebraic signature $\Sigma = \langle S, \Omega \rangle$ and two S-sorted sets X and Y, show that their disjoint union $X \uplus Y$ is the coproduct of X and Y in the category $\mathbf{T}_\Sigma$ of substitutions over Σ (recall Example 3.1.13), where the coproduct injections are the identity substitutions (of the corresponding variables from $X \uplus Y$ for variables in X and in Y, respectively). Generalise this to the category $\mathbf{T}_\Sigma/\Phi$ of substitutions over Σ modulo a set Φ of Σ-equations (cf. Exercise 3.1.14). Finally, characterise coproducts in the category $\mathbf{T}_{\Sigma,\Phi}$, the algebraic theory over Σ generated by Φ (Exercise 3.1.15). □

3.2.3 Equalisers and coequalisers

We have defined above products and coproducts for arbitrary pairs of objects in a category. In this section we deal with constructions for pairs of morphisms constrained to be *parallel*, i.e., pairs of morphisms that have the same source and the same target.

Definition 3.2.19 (Equaliser). An *equaliser* of two parallel morphisms $f: A \to B$ and $g: A \to B$ is an object $E \in |\mathbf{K}|$ together with a morphism $h: E \to A$ such that $h; f = h; g$, and such that for any object $E' \in |\mathbf{K}|$ and morphism $h': E' \to A$ satisfying $h'; f = h'; g$ there is exactly one morphism $k: E' \to E$ such that $k; h = h'$:



□

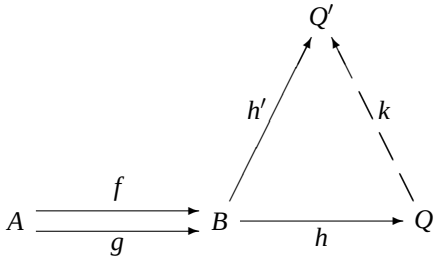
Exercise 3.2.20. Show that an equaliser of $f: A \to B$ and $g: A \to B$ is unique up to isomorphism. □

Exercise 3.2.21. Show that every equaliser (to be more precise: its morphism part) is mono, and every epi equaliser is iso. □

Exercise 3.2.22. Construct equalisers of pairs of parallel morphisms in **Set**. Then, for any signature Σ, construct equalisers of pairs of parallel morphisms in $\mathbf{Alg}(\Sigma)$. HINT: For any two functions $f, g: A \to B$ consider the set $\{a \in A \mid f(a) = g(a)\} \subseteq A$. □

3.2.3.1 Dually:

Definition 3.2.23 (Coequaliser). The dual notion to equaliser is *coequaliser*. The diagram now looks as follows:



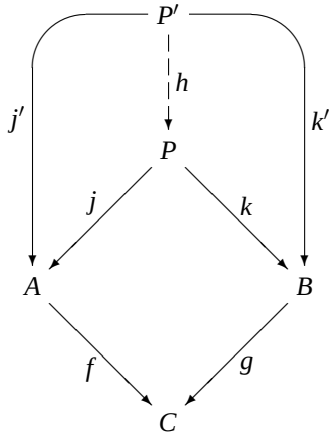
Exercise. Formulate explicitly the definition of a coequaliser. Then dualise the exercises for equalisers. □

Exercise 3.2.24. What is the coequaliser of two morphisms in **Set**? What is the coequaliser of two morphisms in **AlgSig**? What is the coequaliser of two morphisms in $\mathbf{Alg}(\Sigma)$? HINT: Given two functions $f,g:A\to B$ consider the quotient of B by the least equivalence relation $\equiv$ on B such that for all $a\in A$, $f(a)\equiv g(a)$. □

Exercise 3.2.25. What is the coequaliser of two morphisms in the category of substitutions $\mathbf{T}_\Sigma$? □

3.2.4 Pullbacks and pushouts

Definition 3.2.26 (Pullback). A *pullback* of two morphisms $f:A\to C$ and $g:B\to C$ having the same codomain is an object $P\in|\mathbf{K}|$ together with a pair of morphisms $j:P\to A$ and $k:P\to B$ such that $j;f=k;g$, and such that for any object $P'\in|\mathbf{K}|$ and pair of morphisms $j':P'\to A$ and $k':P'\to B$ satisfying $j';f=k';g$ there is exactly one morphism $h:P'\to P$ such that the following diagram commutes:



□

Exercise 3.2.27. Show that a pullback of $f:A\to C$ and $g:B\to C$ is unique up to isomorphism. □

Exercise 3.2.28. Show that if **K** has products (of all pairs of objects) and equalisers (of all pairs of parallel morphisms) than it has pullbacks as well (i.e., all pairs of morphisms with common target have pullbacks in **K**).

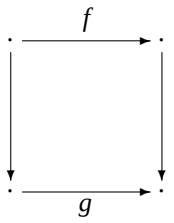
HINT: To construct a pullback of $f:A \to C$ and $g:B \to C$, first construct the product $A \times B$ with projections $\pi_A:A \times B \to A$ and $\pi_B:A \times B \to B$ and then the equaliser $h:P \to A \times B$ of $\pi_A;f:A \times B \to C$ and $\pi_B;g:A \times B \to C$. □

Exercise 3.2.29. Construct the pullback of two morphisms in **Set**, then in **Alg**(Σ), and in **AlgSig**. □

Exercise 3.2.30. Prove that if **K** has a terminal object and all pullbacks (i.e., any pair of **K**-morphisms with common target has a pullback in **K**) then:

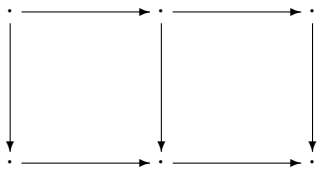
1. **K** has all (binary) products.
2. **K** has all equalisers. HINT: Get the equaliser of $f,g:A \to B$ from the pullback of $\langle id_A, f\rangle, \langle id_A, g\rangle:A \to A \times B$. □

Exercise 3.2.31. Prove that pullbacks translate monomorphisms to monomorphisms: if



is a pullback square and g is mono, then f is mono as well. □

Exercise 3.2.32. Consider the following diagram:

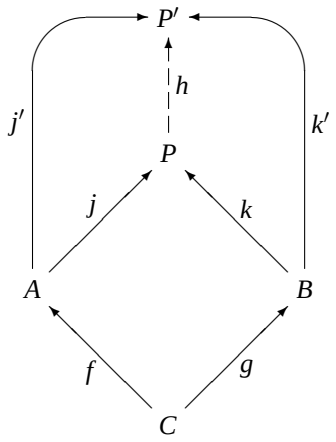


Prove that:

1. If the two squares are pullbacks then the outer rectangle is a pullback.
2. If the diagram commutes and the outer rectangle and right-hand square are both pullbacks then so is the left-hand square. □

3.2.4.1 Dually:

Definition 3.2.33 (Pushout). The dual notion to pullback is *pushout*. The diagram now looks as follows:



Exercise. Spell out the definition of a pushout explicitly. Then dualise the exercises for pullbacks. □

Pushouts provide a basic tool for "putting together" structures of different kinds. Given two objects A and B, a pair of morphisms $f{:}C \to A$ and $g{:}C \to B$ indicates a common source from which some "parts" of A and B come. The pushout of f and g puts together A and B while identifying the parts coming from the common source as indicated by f and g, but keeping the new parts disjoint (cf. the dual of Exercise 3.2.28).

Example 3.2.34. Working in **Set**, consider:

$$\begin{array}{ll} A = \{1,2,3\} & \\ B = \{3,4,5\} & \\ C = \{\clubsuit\} & \\ f = \{\clubsuit \mapsto 2\} & :C \to A \\ g = \{\clubsuit \mapsto 4\} & :C \to B \end{array}$$

Then the pushout object P is (up to isomorphism) given as follows:

$$\begin{array}{ll} P = \{1', \{2'{=}4''\}, 3', 3'', 5''\} & \\ j = \{1 \mapsto 1', 2 \mapsto \{2'{=}4''\}, 3 \mapsto 3'\} & :A \to P \\ k = \{3 \mapsto 3'', 4 \mapsto \{2'{=}4''\}, 5 \mapsto 5''\} & :B \to P \end{array}$$ □

Example 3.2.35. The general comments above about the use of pushouts for putting together objects in categories apply in particular when one wants to combine algebraic signatures, as we will frequently do throughout the rest of the book. As a very simple example of a pushout in the category **AlgSig** of algebraic signatures, consider the signature $\Sigma\mathrm{NAT}$ of natural numbers defined in Exercise 2.5.4. Then, let $\Sigma\mathrm{NAT}_{fib}$ be its extension by a new operation name $fib{:}\,nat \to nat$ and $\Sigma\mathrm{NAT}_{mult}$ its extension by another operation name $mult{:}\,nat \times nat \to nat$. We then have two signature inclusions:

$$\Sigma\mathrm{NAT}_{fib} \hookleftarrow \Sigma\mathrm{NAT} \hookrightarrow \Sigma\mathrm{NAT}_{mult}$$

Their pushout in **AlgSig** yields a signature $\Sigma\mathrm{NAT}_{fib,mult}$ which (up to isomorphism) consists of the shared signature $\Sigma\mathrm{NAT}$ (once, no repetitions!) together with each of the operations added by the two extensions.

This is deceptively simple though, involving only single-sorted signature inclusions that introduce different operation names.

Exercise. Give examples of pushouts in **AlgSig** with signatures involving more than one sort, operation names that coincide, and signature morphisms that are not injective on sorts and/or on operation names. □

3.2.5 The general situation

The definitions introduced in the previous subsections followed a common, more general pattern. As an example, let's have another look at the definition of a pullback (Definition 3.2.26; the notation below refers to the diagram there). Given a diagram in the category at hand (the two morphisms f and g of which we construct the pullback), we consider an object P in this category together with morphisms going from this object to the nodes of the diagram (j, k and an anonymous $c\colon P \to C$) such that all the resulting paths starting from P commute ($j;f = c = k;g$ — hence c may remain anonymous). Moreover, from among all such objects we choose the one that is in a sense "closest" to the diagram: for any object P' with morphisms from it to the diagram nodes (j', k' and an anonymous c') satisfying the required commutativity property ($j';f = c' = k';g$), P' may be uniquely projected onto the chosen object P (via a morphism h) so that all the resulting paths starting from P' commute ($h;j = j'$ and $h;k = k'$, which also implies $h;c = c'$). This is usually referred to as the *universal property* of pullbacks and, more generally, of arbitrary *limits* as defined below. The (dual) universal property of pushouts and, more generally, of arbitrary *colimits* as defined below, may be described by looking at objects with morphisms going from the nodes of a diagram into them. We will formalise this in the rest of this section.

Definition 3.2.36 (Graph). Let Σ_G be the following signature:

sorts *node*, *edge*
ops $source\colon edge \to node$
$target\colon edge \to node$

A Σ_G-algebra is called a *graph*. (Note that these graphs may have multiple edges between any two nodes; such graphs are sometimes called *multigraphs*.) The category **Graph** of graphs is $\mathbf{Alg}(\Sigma_G)$. Given a graph G, we write $e\colon n \to m$ as an abbreviation for $n, m \in |G|_{node}$, $e \in |G|_{edge}$, $source_G(e) = n$ and $target_G(e) = m$. □

Exercise 3.2.37. Construct an initial object, coproducts, coequalisers and pushouts in **Graph**. □

Exercise 3.2.38. Define formally the category $\mathbf{Path}(G)$ of paths in a graph G, where:

Objects of $\mathbf{Path}(G)$: $|G|_{node}$;
Morphisms of $\mathbf{Path}(G)$: paths in G, i.e., finite sequences $e_1 \dots e_n$ of elements of $|G|_{edge}$ such that $source_G(e_{i+1}) = target_G(e_i)$ for $i < n$. Notice that we have to allow for $n = 0$. □

A diagram in **K** is a graph having nodes labelled with **K**-objects and edges labelled with **K**-morphisms with the appropriate source and target. Formally:

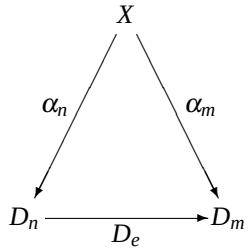
Definition 3.2.39 (Diagram). A *diagram* D in **K** consists of:

- a graph $G(D)$;
- for each node $n \in |G(D)|_{node}$, an object $D_n \in |\mathbf{K}|$; and
- for each edge $e: n \to m$ in $G(D)$, a morphism $D_e: D_n \to D_m$.

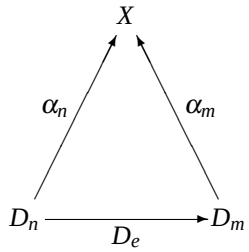
A diagram D is *connected* if its graph $G(D)$ is connected (that is, any two nodes in $G(D)$ are linked by a sequence of edges disregarding their direction, or fully formally: if the total relation on the set of nodes of $G(D)$ is the only equivalence between the nodes that links all nodes having an edge between them). □

Exercise 3.2.40. Show how every small category **K** gives rise to a graph $G(\mathbf{K})$ and a diagram $D(\mathbf{K})$. □

Definition 3.2.41 (Cone and cocone). A *cone* α *over a diagram* D *in* **K** is a **K**-object X together with a family of **K**-morphisms $\langle \alpha_n: X \to D_n \rangle_{n \in |G(D)|_{node}}$ such that for every edge $e: n \to m$ in the graph $G(D)$ the following diagram commutes:



Dually: A *cocone* α *over a diagram* D *in* **K** is a **K**-object X together with a family of **K**-morphisms $\langle \alpha_n: D_n \to X \rangle_{n \in |G(D)|_{node}}$ such that for every edge $e: n \to m$ in the graph $G(D)$ the following diagram commutes:

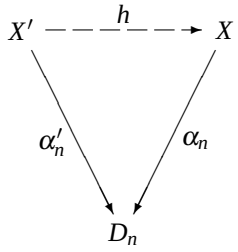


□

In the following we will write cones simply as families $\langle \alpha_n : X \to D_n \rangle_{n \in |G(D)|_{node}}$, omitting any explicit mention of the apex X, and similarly for cocones. The notation is not quite justified only in the case when the diagram (and hence the family) is empty; this will not lead to any misunderstanding.

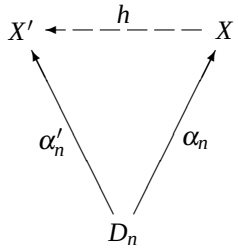
Let D be a diagram in $\mathbf{K}$ with $|G(D)|_{node} = N$ and $|G(D)|_{edge} = E$.

Definition 3.2.42 (Limit and colimit). A *limit of D in* $\mathbf{K}$ is a cone $\langle \alpha_n : X \to D_n \rangle_{n \in N}$ such that for any cone $\langle \alpha'_n : X' \to D_n \rangle_{n \in N}$ there is exactly one morphism $h : X' \to X$ such that for every $n \in N$ the following diagram commutes:



If $\langle \alpha_n : X \to D_n \rangle_{n \in N}$ is a limit of D, we will refer to X as the *limit object* of D (or sometimes just the *limit* of D), and to the morphisms α_n, $n \in N$, as the limit *projections*.

Dually: A *colimit of D in* $\mathbf{K}$ is a cocone $\langle \alpha_n : D_n \to X \rangle_{n \in N}$ such that for any cocone $\langle \alpha'_n : D_n \to X' \rangle_{n \in N}$ there is exactly one morphism $h : X \to X'$ such that for every $n \in N$ the following diagram commutes:



If $\langle \alpha_n : D_n \to X \rangle_{n \in N}$ is a colimit of D, we will refer to X as the *colimit object* of D (or sometimes just the *colimit* of D), and to the morphisms α_n, $n \in N$, as the colimit *injections*. □

Definition 3.2.43 (Completeness and cocompleteness). A category $\mathbf{K}$ is *(finitely) complete* if every (finite) diagram in $\mathbf{K}$ has a limit. Dually, $\mathbf{K}$ is *(finitely) cocomplete* if every (finite) diagram in $\mathbf{K}$ has a colimit. □

Exercise 3.2.44. Define formally the category $\mathbf{Cone}(D)$ of cones over a diagram D, where:

Objects of $\mathbf{Cone}(D)$: cones over D;
Morphisms of $\mathbf{Cone}(D)$: a morphism from $\alpha = \langle \alpha_n : X \to D_n \rangle_{n \in N}$ to $\alpha' = \langle \alpha'_n : X' \to D_n \rangle_{n \in N}$ is a $\mathbf{K}$-morphism $h : X \to X'$ such that $\alpha_n = h ; \alpha'_n$ for $n \in N$.

Prove that the limit of D is a terminal object in $\mathbf{Cone}(D)$. Note that this implies that a limit of any diagram is unique up to isomorphism.

Present the category of objects over an object (cf. Definition 3.1.28) as the category of cones over a certain diagram. □

Exercise 3.2.45. Show that products, terminal objects, equalisers and pullbacks in $\mathbf{K}$ are limits of simple diagrams in $\mathbf{K}$. □

Exercise 3.2.46. Construct in **Set** a limit of the diagram

$$A_0 \xleftarrow{f_0} A_1 \xleftarrow{f_1} A_2 \xleftarrow{f_2} A_3 \xleftarrow{f_3} \cdots$$ □

Exercise 3.2.47. Show that limiting cones are *jointly mono*: if $\langle \alpha_n : X \to D_n \rangle_{n \in |G(D)|_{node}}$ is a limit of D, then $f = g$ whenever for all $n \in |G(D)|_{node}$, $f;\alpha_n = g;\alpha_n$. □

Exercise 3.2.48. Show that if $\mathbf{K}$ has a terminal object, binary products and all equalisers then it is finitely complete. HINT: Given a finite diagram in $\mathbf{K}$, first build the product of all its objects, and then gradually turn it into a limit by "equalising" the triangles formed by product projections and morphisms in the diagram.

Use Exercise 3.2.30 to conclude that if $\mathbf{K}$ has a terminal object and all pullbacks then it is finitely complete. □

Exercise 3.2.49. Show that if $\mathbf{K}$ has products of arbitrary families of objects and all equalisers then it is complete. HINT: Proceed as in Exercise 3.2.48, but notice that all the triangles involved may be "equalised" simultaneously in one step, cf. [Mac71], Theorem V.2.1. □

Exercise 3.2.50. A *wide pullback* is the limit of a non-empty family of morphisms with a common target. Show that if a category has a terminal object and all wide pullbacks then it has products of arbitrary families of objects, and then conclude that it is complete. HINT: Generalise Exercise 3.2.30 and use Exercise 3.2.49. □

Exercise 3.2.51. Recall that for any category $\mathbf{K}$ and object $A \in |\mathbf{K}|$, $\mathbf{K}{\downarrow}A$ is the slice category of objects over A (Definition 3.1.28).

Notice that $\mathbf{K}{\downarrow}A$ has a terminal object. Then show that binary products in $\mathbf{K}{\downarrow}A$ are essentially given by the pullbacks in $\mathbf{K}$ (of morphisms to A) and similarly, arbitrary non-empty products in $\mathbf{K}{\downarrow}A$ are essentially given by wide pullbacks in $\mathbf{K}$. Check also that any (wide) pullback in $\mathbf{K}{\downarrow}A$ is given by the corresponding (wide) pullback in $\mathbf{K}$ (no morphisms to A added).

Conclude that $\mathbf{K}{\downarrow}A$ is finitely complete if $\mathbf{K}$ has all pullbacks, and $\mathbf{K}{\downarrow}A$ is complete if $\mathbf{K}$ has all wide pullbacks. □

Exercise 3.2.52. Dualise the above exercises. □

Exercise 3.2.53. Show that:

1. **Set** is complete and cocomplete.

2. **FinSet** is finitely complete and finitely cocomplete, but is neither complete nor cocomplete.
3. **Alg**(Σ) is complete for any signature Σ. (It is also cocomplete, but the proof is harder — give it a try!)
4. **AlgSig** is cocomplete. (Is it complete?)

HINT: Use Exercise 3.2.49 and its dual, and the standard constructions of (co)products and (co)equalisers in these categories hinted at in Examples 3.2.9, 3.2.16 and Exercises 3.2.22, 3.2.24. Check that, given a diagram D with nodes N and edges E in **Set**, its limit is (up to isomorphism) the set of families $\langle d_n \rangle_{n \in N}$ that are compatible with D in the sense that $d_n \in D_n$ for each $n \in N$ and $d_m = D_e(d_n)$ for each edge $e: n \to m$, with the obvious projections. Check that its colimit is (up to isomorphism) the quotient of the disjoint union $\biguplus_{n \in N} D_n$ by the least equivalence relation that is generated by all pairs $\langle d_n, D_e(d_n) \rangle$ for $e: n \to m$ in E and $d_n \in D_n$. □

Exercise 3.2.54. Show that $\mathbf{AlgSig}^{der}$ is not finitely cocomplete. (HINT: Consider a morphism mapping a binary operation to the projection on the first argument and another morphism mapping the same operation to the projection on the second argument. Can such a pair of morphisms have a coequaliser?) □

Exercise 3.2.55. When is a preorder category (finitely) complete and cocomplete? □

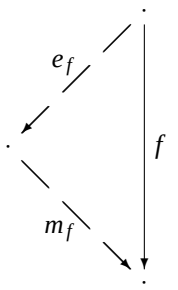
3.3 Factorisation systems

In this section we will interrupt our presentation of the basic concepts of category theory and try to illustrate how they can be used to formulate some well-known ideas at a level of generality and abstraction that ensures their applicability in many specific contexts.

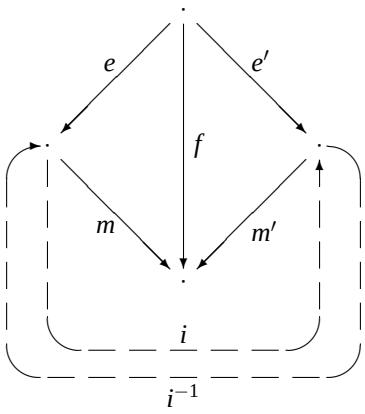
The concept on which we concentrate here is that of *reachability* (cf. Section 1.2). Recall that the original definition of a reachable algebra used the notion of a subalgebra (cf. Definition 1.2.7). Keeping in mind that in the categorical framework we deal with objects identified up to isomorphism, we slightly generalise the standard formulation and, for any signature $\Sigma \in |\mathbf{AlgSig}|$, say that a Σ-algebra B is a subalgebra of A if there exists an *injective* Σ-homomorphism from B to A. A dual notion is that of a *quotient*: a Σ-algebra B is a quotient of a Σ-algebra A if there exists a *surjective* Σ-homomorphism from A to B. Now, a Σ-algebra A is *reachable* if it has no proper subalgebra (i.e., every subalgebra of A is isomorphic to A), or equivalently, if it is a quotient of the algebra T_Σ of ground Σ-terms (cf. Exercise 1.4.14). In this formulation, the above definitions may be used to introduce a notion of reachability in an arbitrary category. However, we need an appropriate generalisation of the concept of injective and surjective homomorphisms. A first attempt might be to use arbitrary epimorphisms and monomorphisms for this purpose, but it soon turns out that these concepts are not "fine enough" to ensure the properties we are after. An appropriate refinement of these is given if the category is equipped with a *factorisation system*.

Definition 3.3.1 (Factorisation system). Let $\mathbf{K}$ be an arbitrary category. A *factorisation system* for $\mathbf{K}$ is a pair $\langle \mathbf{E}, \mathbf{M} \rangle$, where:

- $\mathbf{E}$ is a collection of epimorphisms in $\mathbf{K}$ and $\mathbf{M}$ is a collection of monomorphisms in $\mathbf{K}$;
- each of $\mathbf{E}$ and $\mathbf{M}$ is closed under composition and contains all isomorphisms in $\mathbf{K}$;
- every morphism in $\mathbf{K}$ has an $\langle \mathbf{E}, \mathbf{M} \rangle$-factorisation: for each $f \in \mathbf{K}$, $f = e_f ; m_f$ for some $e_f \in \mathbf{E}$ and $m_f \in \mathbf{K}$;



- $\langle \mathbf{E}, \mathbf{M} \rangle$-factorisations are unique up to isomorphism: for any $e, e' \in \mathbf{E}$ and $m, m' \in \mathbf{M}$, if $e ; m = e' ; m'$ then there exists an isomorphism i such that $e ; i = e'$ and $i ; m' = m$.



□

Example 3.3.2. Set has a factorisation system $\langle \mathbf{E}, \mathbf{M} \rangle$, where $\mathbf{E}$ is the collection of all surjective functions and $\mathbf{M}$ is the collection of all injective functions. □

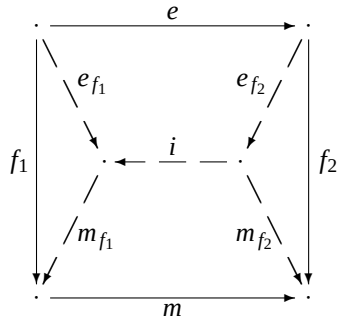
Example 3.3.3. For any signature Σ, $\mathbf{Alg}(\Sigma)$ has a factorisation system[4] $\langle \mathbf{TE}_\Sigma, \mathbf{TM}_\Sigma \rangle$, where $\mathbf{TE}_\Sigma$ is the collection of all surjective Σ-homomorphisms and $\mathbf{TM}_\Sigma$ is the collection of all injective Σ-homomorphisms; see Exercise 1.3.23. □

Consider an arbitrary category $\mathbf{K}$ equipped with a factorisation system $\langle \mathbf{E}, \mathbf{M} \rangle$.

[4] "**T**" in $\mathbf{TE}_\Sigma$ and $\mathbf{TM}_\Sigma$ indicates that we are dealing with ordinary *total* algebras here, as opposed to partial and continuous algebras with the factorisation systems discussed below.

Lemma 3.3.4 (Diagonal fill-in lemma). *For any morphisms f_1, f_2, e, m in* **K**, *where $e \in \mathbf{E}$ and $m \in \mathbf{M}$, if $f_1;m = e;f_2$ then there exists a unique morphism g such that $e;g = f_1$ and $g;m = f_2$.*

Proof sketch. The required "diagonal" is given by $g = e_{f_2};i;m_{f1}$, as illustrated by the diagram below; its uniqueness follows easily since e is an epimorphism.



□

Exercise 3.3.5. Show that if $e \in \mathbf{E}$ and $e;f \in \mathbf{M}$ for some morphism $f \in \mathbf{K}$, then e is an isomorphism. Dually, if $m \in \mathbf{M}$ and $f;m \in \mathbf{E}$ for some morphism $f \in \mathbf{K}$, then m is an isomorphism. □

Definition 3.3.6 (Subobject and quotient). Let $A \in |\mathbf{K}|$. A *subobject* of A is an object $B \in |\mathbf{K}|$ together with a morphism $m\colon B \to A$ such that $m \in \mathbf{M}$. A *quotient* of A is an object $B \in |\mathbf{K}|$ together with a morphism $e\colon A \to B$ such that $e \in \mathbf{E}$. □

Definition 3.3.7 (Reachable object). An object $A \in |\mathbf{K}|$ is *reachable* if it has no proper subobject, i.e., if every morphism $m \in \mathbf{M}$ with target A is an isomorphism.

□

The category $\mathbf{Alg}(\Sigma)$ of Σ-algebras and the notion of a reachable algebra provide an instance of the general concept of reachability introduced in the above definition. The following theorem gives more general versions of well-known facts often laboriously proved in the standard algebraic framework.

Theorem 3.3.8. *Assume that* **K** *has an initial object Λ. Then:*

1. *An object $A \in |\mathbf{K}|$ is reachable iff it is a quotient of the initial object Λ.*
2. *Every object in $|\mathbf{K}|$ has a reachable subobject which is unique up to isomorphism.*
3. *If $A \in |\mathbf{K}|$ is reachable then for every $B \in |\mathbf{K}|$ there exists at most one morphism from A to B.*
4. *If $A \in |\mathbf{K}|$ is reachable and $f \in \mathbf{K}$ is a morphism with target A then $f \in \mathbf{E}$.* □

Exercise 3.3.9. Prove the theorem and identify the familiar facts about reachable algebras generalised here. □

One of the main results of Chapter 2, Theorem 2.5.14, states that any equational specification has an initial model. This is just a special case of a more general result which we formulate and prove for an arbitrary category with "reachability structure" satisfying an additional, technical property that any object has up to isomorphism only a *set* of quotients.

Definition 3.3.10 (Co-well-powered category). $\mathbf{K}$ is $\mathbf{E}$-*co-well-powered* if for any $A \in |\mathbf{K}|$ there exists a *set* of morphisms $E \subseteq \mathbf{E}$ such that for every morphism $e \in \mathbf{E}$ with source A there exist a morphism $e' \in E$ and an isomorphism i such that $e = e';i$. □

Definition 3.3.11 (Quasi-variety). A collection of objects $Q \subseteq |\mathbf{K}|$ is a *quasi-variety* if it is closed under subobjects and products of non-empty sets of objects in Q. □

Lemma 3.3.12 (Initiality lemma). *Assume that* $\mathbf{K}$ *has an initial object, is* $\mathbf{E}$*-co-well-powered, and any set of objects in* $\mathbf{K}$ *has a product. Then any non-empty quasi-variety in* $\mathbf{K}$ *(considered as the corresponding full subcategory of* $\mathbf{K}$*) has an initial object which is reachable in* $\mathbf{K}$*.*

Proof. Let $Q \subseteq |\mathbf{K}|$ be a non-empty collection of objects closed under subobjects and products of non-empty sets. Let Q_r be a *set* of reachable objects in Q such that every reachable object in Q is isomorphic to an element of Q_r (such a set exists since $\mathbf{K}$ is $\mathbf{E}$-co-well-powered). The reachable subobject of the product of Q_r (which is unique up to isomorphism) is a reachable initial object in Q. □

It is now easy to check that in the context of Example 3.3.3 every class of Σ-algebras definable by a set of Σ-equations is a non-empty quasi-variety, and hence Lemma 3.3.12 indeed directly implies Theorem 2.5.14.

We conclude this section with two examples of categories naturally equipped with a notion of reachability which is an instance of the general concept introduced above.

Example 3.3.13. Recall Definitions 2.7.30 and 2.7.31 of partial Σ-algebras and Σ-homomorphisms between them. For any signature Σ, define the category of partial Σ-algebras, $\mathbf{PAlg}(\Sigma)$, as follows:

Objects of $\mathbf{PAlg}(\Sigma)$: partial Σ-algebras;
Morphisms of $\mathbf{PAlg}(\Sigma)$: weak Σ-homomorphisms.

Define also the subcategory $\mathbf{PAlg}_{\mathbf{str}}(\Sigma)$ of partial Σ-algebras with *strong* homomorphisms between them, as follows:

Objects of $\mathbf{PAlg}_{\mathbf{str}}(\Sigma)$: partial Σ-algebras;
Morphisms of $\mathbf{PAlg}_{\mathbf{str}}(\Sigma)$: strong Σ-homomorphisms.

The category $\mathbf{PAlg}(\Sigma)$ of partial Σ-algebras has a factorisation system $\langle \mathbf{PE}_\Sigma, \mathbf{PM}_\Sigma \rangle$, where $\mathbf{PE}_\Sigma$ is the collection of all epimorphisms in $\mathbf{PAlg}(\Sigma)$ and $\mathbf{PM}_\Sigma$ is the collection of all monomorphisms in $\mathbf{PAlg}(\Sigma)$ that are strong Σ-homomorphisms.

Exercise. Characterise epimorphisms in $\mathbf{PAlg}(\Sigma)$ (they are not surjective in general) and prove that $\langle \mathbf{PE}_\Sigma, \mathbf{PM}_\Sigma \rangle$ is indeed a factorisation system for $\mathbf{PAlg}(\Sigma)$. Check then that factorisation of a strong Σ-homomorphism in $\langle \mathbf{PE}_\Sigma, \mathbf{PM}_\Sigma \rangle$ consists of strong Σ-homomorphisms. Conclude that strong homomorphisms in $\mathbf{PE}_\Sigma$ and $\mathbf{PM}_\Sigma$, respectively, form a factorization system for $\mathbf{PAlg}_{\mathbf{str}}(\Sigma)$. □

Example 3.3.14. For any signature Σ, define the category of continuous Σ-algebras, $\mathbf{CAlg}(\Sigma)$, as follows:

Objects of $\mathbf{CAlg}(\Sigma)$: *continuous Σ-algebras*, which are just like ordinary (total) Σ-algebras, except that their carriers are required to be complete partial orders and their operations are continuous functions (cf. Exercise 3.1.9);

Morphisms of $\mathbf{CAlg}(\Sigma)$: *continuous Σ-homomorphisms*: a continuous Σ-homomorphism from a continuous Σ-algebra A to a continuous Σ-algebra B is a Σ-homomorphism $h:A \to B$ which is continuous as a function between complete partial orders. We say that h is *full* if it reflects the ordering, i.e., for all $a, a' \in |A|_s$, $h(a) \leq_B h(a')$ implies $a \leq_A a'$.

The category $\mathbf{CAlg}(\Sigma)$ of continuous Σ-algebras has a factorisation system $\langle \mathbf{CE}_\Sigma, \mathbf{CM}_\Sigma \rangle$, where $\mathbf{CM}_\Sigma$ is the collection of all full monomorphisms in $\mathbf{CAlg}(\Sigma)$ and $\mathbf{CE}_\Sigma$ is the collection of all *strongly dense* epimorphisms in $\mathbf{CAlg}(\Sigma)$. A continuous Σ-homomorphism $h:A \to B$ is strongly dense if B has no proper continuous subalgebra which contains the set-theoretic image of $|A|$ under h. (Note that the expected notion of a continuous subalgebra is determined by the chosen collection of factorisation monomorphisms $\mathbf{CM}_\Sigma$.) This is equivalent to the requirement that every element of $|B|$ is the least upper bound of a countable chain of least upper bounds of countable chains of $\ldots$ of elements in the set-theoretic image of $|A|$ under h. Consequently, given a strongly dense continuous homomorphism $h:A \to B$, every element of $|B|$ is the least upper bound of a subset (not necessarily a chain though) of the set-theoretic image of $|A|$ under h, which yields the key argument to show that $\mathbf{CAlg}(\Sigma)$ is $\mathbf{CE}_\Sigma$-co-well-powered.

Exercise. Prove that $\langle \mathbf{CE}_\Sigma, \mathbf{CM}_\Sigma \rangle$ is indeed a factorisation system for $\mathbf{CAlg}(\Sigma)$. Also, try to construct an example of an epimorphism in $\mathbf{CAlg}(\Sigma)$ which is not strongly dense. □

Exercise 3.3.15. Characterise reachable algebras in $\mathbf{PAlg}(\Sigma)$ and in $\mathbf{CAlg}(\Sigma)$. Instantiate the facts listed in Theorem 3.3.8 to these categories. □

3.4 Functors and natural transformations

As explained in the introduction to this chapter, for category theorists it is tantamount to heresy to consider objects in the absence of morphisms between them. Up to now we have departed from this dogma in our study of categories themselves; in the previous sections of this chapter we have worked with categories without introducing any notion of a morphism between them. We hasten here to correct this lapse: morphisms between categories are *functors*, to be introduced in this section. And by way of atonement we will also introduce *natural transformations*, which are morphisms between functors.

3.4.1 Functors

A category consists of a collection of objects and a collection of morphisms with structure given by the choice of sources and targets of morphism, by the definition of composition and by the identities that are assumed to exist. As in other standard cases of collections with additional structure, morphisms between categories are maps between the collections of objects and morphisms, respectively, that preserve this structure.

Definition 3.4.1 (Functor). A *functor* $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ from a category **K1** to a category **K2** consists of:

- a function $\mathbf{F}_{Obj}:|\mathbf{K1}| \to |\mathbf{K2}|$; and
- for each $A,B \in |\mathbf{K1}|$, a function $\mathbf{F}_{A,B}:\mathbf{K1}(A,B) \to \mathbf{K2}(\mathbf{F}_{Obj}(A),\mathbf{F}_{Obj}(B))$

such that:

- **F** preserves identities: $\mathbf{F}_{A,A}(id_A) = id_{\mathbf{F}_{Obj}(A)}$ for all objects $A \in |\mathbf{K}|$; and
- **F** preserves composition: for all morphisms $f : A \to B$ and $g : B \to C$ in **K1**, $\mathbf{F}_{A,C}(f;g) = \mathbf{F}_{A,B}(f);\mathbf{F}_{B,C}(g)$. □

Notation. We use **F** to refer to both $\mathbf{F}_{Obj}$ and $\mathbf{F}_{A,B}$ for all $A,B \in |\mathbf{K1}|$. □

In the literature, functors as defined above are sometimes referred to as *covariant* functors. A *contravariant* functor is then defined in the same way except that it "reverses the direction of morphisms", i.e., a contravariant functor $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ maps a **K1**-morphism $f:A \to B$ to a **K2**-morphism $\mathbf{F}(f):\mathbf{F}(B) \to \mathbf{F}(A)$. Even though we will use this terminology sometimes, no new formal definition is required: a contravariant functor from **K1** to **K2** is a (covariant) functor from $\mathbf{K1}^{op}$ to **K2** (cf. e.g. Examples 3.4.7 and 3.4.29 below).

Example 3.4.2 (Identity functor). A functor $\mathbf{Id_K}:\mathbf{K} \to \mathbf{K}$ is defined in the obvious way. □

Example 3.4.3 (Inclusion functor). If **K1** is a subcategory of **K2** then the inclusion $\mathbf{I}:\mathbf{K1} \hookrightarrow \mathbf{K2}$ is a functor. □

Example 3.4.4 (Constant functor). For any $A \in |\mathbf{K2}|$, $\mathbf{C}_A:\mathbf{K1} \to \mathbf{K2}$ is a functor, where $\mathbf{C}_A(B) = A$ for any $B \in |\mathbf{K1}|$ and $\mathbf{C}_A(f) = id_A$ for any **K1**-morphism f. □

Example 3.4.5 (Opposite functor). For any functor $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$, there is a functor $\mathbf{F}^{op}:\mathbf{K1}^{op} \to \mathbf{K2}^{op}$ which is the "same" as **F**, but is considered between the opposite categories. □

Example 3.4.6 (Powerset functor). $\mathcal{P}:\mathbf{Set} \to \mathbf{Set}$ is a functor, where $\mathcal{P}(X) = \{Y \mid Y \subseteq X\}$ for any set X, and for any function $f:X \to X'$, $\mathcal{P}(f):\mathcal{P}(X) \to \mathcal{P}(X')$ is defined by $\mathcal{P}(f)(Y) = \{f(y) \mid y \in Y\}$. □

Example 3.4.7 (Contravariant powerset functor). $\mathcal{P}_{-1}\colon \mathbf{Set}^{op} \to \mathbf{Set}$ is a functor, where $\mathcal{P}_{-1}(X) = \{Y \mid Y \subseteq X\}$ for any set X, and for any morphism $f\colon X \to X'$ in $\mathbf{Set}^{op}$ (i.e., any function $f\colon X' \to X$), $\mathcal{P}_{-1}(f)\colon \mathcal{P}_{-1}(X) \to \mathcal{P}_{-1}(X')$ is defined by $\mathcal{P}_{-1}(f)(Y) = \{x' \in X' \mid f(x') \in Y\}$. □

Example 3.4.8 (Sequence functor). $\mathbf{Seq}\colon \mathbf{Set} \to \mathbf{Mon}$ is a functor, where **Mon** is the category of monoids with monoid homomorphisms as morphisms. For any set $X \in |\mathbf{Set}|$, $\mathbf{Seq}(X) = \langle X^*, \hat{}, \varepsilon \rangle$, where X^* is the set of all finite sequences of elements from X, $\hat{}$ is sequence concatenation, and ε is the empty sequence. Then, for any function $f\colon X \to Y$, $\mathbf{Seq}(f)\colon \mathbf{Seq}(X) \to \mathbf{Seq}(Y)$ is the homomorphism defined by $\mathbf{Seq}(f)(x_1 \ldots x_n) = f(x_1) \ldots f(x_n)$. □

Example 3.4.9 (Reduct functor). For any signature morphism $\sigma\colon \Sigma \to \Sigma'$, $\_|_\sigma\colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$ is a functor that takes each Σ'-algebra A' to its σ-reduct $A'|_\sigma \in |\mathbf{Alg}(\Sigma)|$ and each Σ'-homomorphisms h' to its σ-reduct $h'|_\sigma$ (cf. Definitions 1.5.4 and 1.5.8). □

Example 3.4.10 (Forgetful functor). Let $\Sigma = \langle S, \Omega \rangle$ be a signature. Then $|\_|\colon \mathbf{Alg}(\Sigma) \to \mathbf{Set}^S$ is the functor that takes each Σ-algebra $A \in |\mathbf{Alg}(\Sigma)|$ to its S-sorted carrier set $|A| \in |\mathbf{Set}^S|$ and each Σ-homomorphism to its underlying S-sorted function. (The functor $|\_|$ should really be decorated with a subscript identifying the signature Σ — we hope that leaving it out will not confuse the reader.) These special reduct functors $|\_|$ will be referred to as *forgetful functors*.

More generally, the term "forgetful functor" is used to refer to any functor that, intuitively, forgets the structure of objects in a category, mapping any structured object to its underlying unstructured set of elements. Thus, in addition to examples that exactly fit the above definition (like the functor mapping any monoid to the set of its elements) this also covers examples like the functor that maps any topological space to the set of its points and the functor that forgets the metric of a metric space. □

Example 3.4.11 (Term algebra). For any signature $\Sigma = \langle S, \Omega \rangle$, there is a functor $T_\Sigma\colon \mathbf{Set}^S \to \mathbf{Alg}(\Sigma)$ that maps any S-sorted set X to the term algebra $T_\Sigma(X)$, and any S-sorted function $f\colon X \to Y$ to the unique Σ-homomorphism $f^\#\colon T_\Sigma(X) \to T_\Sigma(Y)$ that extends f. □

Exercise 3.4.12. For any signature Σ and set Φ of Σ-equations, define the *quotient functor* $\_/\Phi\colon \mathbf{Alg}(\Sigma) \to \mathbf{Alg}(\Sigma)$ such that for any Σ-algebra A, A/Φ is the quotient of A by the least congruence $\simeq$ on A generated by Φ, that is, such that $t_A(v) \simeq t'_A(v)$ for each Σ-equation $\forall X \bullet t = t'$ in Φ and valuation $v\colon X \to |A|$. Make sure that what you define is a functor! □

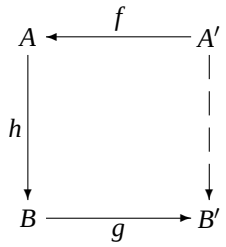
Exercise 3.4.13. For any signature Σ, define the *restriction functor* $\mathbf{R}_\Sigma\colon \mathbf{Alg}(\Sigma) \to \mathbf{Alg}(\Sigma)$ such that for any Σ-algebra A, $\mathbf{R}_\Sigma(A)$ is the reachable subalgebra of A.

More generally: let $\mathbf{K}$ be an arbitrary category with an initial object and a factorisation system, and let $\mathbf{K}_R$ be the full subcategory of $\mathbf{K}$ determined by the collection of all reachable objects in $\mathbf{K}$ (cf. Section 3.3). Define a functor $\mathbf{R}_\mathbf{K}\colon \mathbf{K} \to \mathbf{K}_R$ that maps any $A \in |\mathbf{K}|$ to the (unique up to isomorphism) reachable subobject of A. □

Example 3.4.14 (Projection functor). For any two categories **K1** and **K2**, the projection functors $\Pi_{\mathbf{K1}}\colon \mathbf{K1} \times \mathbf{K2} \to \mathbf{K1}$ and $\Pi_{\mathbf{K2}}\colon \mathbf{K1} \times \mathbf{K2} \to \mathbf{K2}$ are defined by $\Pi_{\mathbf{K1}}(\langle A, B\rangle) = A$ and $\Pi_{\mathbf{K1}}(\langle f, g\rangle) = f$, and $\Pi_{\mathbf{K2}}(\langle A, B\rangle) = B$ and $\Pi_{\mathbf{K2}}(\langle f, g\rangle) = g$. □

Example 3.4.15 (Hom-functor). Let **K** be a locally small category. $\mathbf{Hom}\colon \mathbf{K}^{op} \times \mathbf{K} \to \mathbf{Set}$ is a functor, where $\mathbf{Hom}(\langle A, B\rangle) = \mathbf{K}(A, B)$ and

$$\mathbf{Hom}(\underbrace{\langle f\colon A' \to A, g\colon B \to B'\rangle}_{\in \mathbf{K}^{op} \times \mathbf{K}(\langle A,B\rangle, \langle A',B'\rangle)})(\underbrace{h\colon A \to B}_{\in \mathbf{Hom}(\langle A,B\rangle)}) = \underbrace{f;h;g}_{\in \mathbf{Hom}(\langle A',B'\rangle)} .$$



□

Exercise 3.4.16 (Exponent functor). For any set X define a functor $[\_ \to X]\colon \mathbf{Set}^{op} \to \mathbf{Set}$ mapping any set to the set of all functions from it to X. That is, for any set $Y \in |\mathbf{Set}|$, $[Y \to X]$ is the set of all functions from Y to X and then for any morphism $f\colon Y \to Y'$ in $\mathbf{Set}^{op}$, which is a function $f\colon Y' \to Y$ in **Set**, $[f \to X]\colon [Y \to X] \to [Y' \to X]$ is defined by pre-composition with f as follows: $[f \to X](g) = f;g$. □

Example 3.4.17 (Converting partial functions to total functions). Let **Pfn** be the category of sets with partial functions and let $\mathbf{Set}_\perp$ be the subcategory of **Set** having sets containing a distinguished element $\perp$ as objects and $\perp$-preserving functions as morphisms. Then $\mathbf{Tot}\colon \mathbf{Pfn} \to \mathbf{Set}_\perp$ converts partial functions to total functions by using $\perp$ to represent "undefined" as follows:

- $\mathbf{Tot}(X) = X \uplus \{\perp\}$
- $\mathbf{Tot}(f)(x) = \begin{cases} f(x) & \text{if } f(x) \text{ is defined} \\ \perp & \text{otherwise} \end{cases}$

Exercise. Notice that strictly speaking the above definition is not well-formed: according to the definition of disjoint union, if X is non-empty then $X \not\subseteq X \uplus \{\perp\}$; thus, given a partial function $f\colon X \to Y$, $\mathbf{Tot}(f)$ as defined above need not be a function from $\mathbf{Tot}(X)$ to $\mathbf{Tot}(Y)$. Restate this definition formally, using explicit injections $\iota_1\colon X \to X \uplus \{\perp\}$ and $\iota_2\colon \{\perp\} \to X \uplus \{\perp\}$ for each set X. □

Example 3.4.18 (Converting partial algebras to total algebras). The same "totalisation" idea as used in the above Example 3.4.17 yields a totalisation functor $\mathbf{Tot}_\Sigma\colon \mathbf{PAlg}_{\mathbf{str}}(\Sigma) \to \mathbf{Alg}(\Sigma)$, for each signature Σ, mapping partial Σ-algebras and their strong homomorphisms to total Σ-algebras and their homomorphisms (cf. Definitions 2.7.30 and 2.7.31, and Example 3.3.13).

Let $\Sigma = \langle S, \Omega\rangle \in |\mathbf{AlgSig}|$. $\mathbf{Tot}_\Sigma\colon \mathbf{PAlg}_{\mathbf{str}}(\Sigma) \to \mathbf{Alg}(\Sigma)$ is defined as follows:

- For any partial Σ-algebra $A \in |\mathbf{PAlg}_{\mathbf{str}}(\Sigma)|$, $\mathbf{Tot}_\Sigma(A) \in |\mathbf{Alg}(\Sigma)|$ is the Σ-algebra whose carriers are obtained from the corresponding carriers of A by adding a distinguished element $\bot$, and whose operations are obtained from the operations of A by making the result $\bot$ for arguments on which the latter are undefined, that is:
  - for each sort name $s \in S$, $|\mathbf{Tot}_\Sigma(A)|_s = |A|_s \uplus \{\bot\}$; and
  - for each operation name $f : s_1 \times \ldots \times s_n \to s$ in Σ, $f_{\mathbf{Tot}_\Sigma(A)}$ is the function which yields $\bot$ if any of its arguments is $\bot$, and for $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$,

$$f_{\mathbf{Tot}_\Sigma(A)}(a_1,\ldots,a_n) = \begin{cases} f_A(a_1,\ldots,a_n) & \text{if } f_A(a_1,\ldots,a_n) \text{ is defined} \\ \bot & \text{otherwise.} \end{cases}$$

- For any strong Σ-homomorphism $h : A \to B$ (which is a family of *total* functions between the corresponding carriers of A and B), $\mathbf{Tot}_\Sigma(h) : \mathbf{Tot}_\Sigma(A) \to \mathbf{Tot}_\Sigma(B)$ is (the family of functions in) h extended to map $\bot$ to $\bot$.

Exercise. Check that for any strong Σ-homomorphism $h : A \to B$, $\mathbf{Tot}_\Sigma(h) : \mathbf{Tot}_\Sigma(A) \to \mathbf{Tot}_\Sigma(B)$ is indeed a Σ-homomorphism. Can you extend $\mathbf{Tot}_\Sigma$ to *weak* Σ-homomorphisms between partial algebras? □

Exercise 3.4.19. Do the above functors map monomorphisms to monomorphisms? Do they map epimorphisms to epimorphisms? What about isomorphisms? (Co)limits? (Co)cones? Anything else you can think of? □

Definition 3.4.20 (Diagram translation). Given a functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ and a diagram D in **K1**, the *translation of D by* $\mathbf{F}$ is defined as the diagram $\mathbf{F}(D)$ in **K2** with the same underlying graph as D and with the labels of D translated by $\mathbf{F}$:

- $G(\mathbf{F}(D)) = G(D)$;
- for each $n \in |G(D)|_{node}$, $\mathbf{F}(D)_n = \mathbf{F}(D_n)$; and
- for each $e \in |G(D)|_{edge}$, $\mathbf{F}(D)_e = \mathbf{F}(D_e)$. □

Exercise 3.4.21 (Diagrams as functors). A diagram D in **K** corresponds to a functor from the category $\mathbf{Path}(G(D))$ of paths in the underlying graph of D to **K**. Formalise this. HINT: Given a diagram D, define a functor that maps each path $e_1 \ldots e_n$ in $G(D)$ to $D_{e_1}; \ldots; D_{e_n}$. Do not forget the case where $n = 0$.

Then, anticipating Definition 3.4.27, define the translation of a diagram by a functor in terms of functor composition. □

Definition 3.4.22 (Functor continuity and cocontinuity). A functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ is (*finitely*) *continuous* if it preserves the existing limits of all (finite) diagrams in **K1**, that is, if for any (finite) diagram D in **K1**, **F** maps any limiting cone over D to a limiting cone over $\mathbf{F}(D)$.

A functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ is (*finitely*) *cocontinuous* if it preserves the existing colimits of all (finite) diagrams in **K1**, that is, if for any (finite) diagram D in **K1**, **F** maps any colimiting cocone over D to a colimiting cocone over $\mathbf{F}(D)$. □

Exercise 3.4.23. Assuming that **K1** is (finitely) complete, use Exercise 3.2.49 to show that a functor $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ is (finitely) continuous if and only if it preserves (finite) products and equalisers.

Similarly, show that $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ is finitely continuous if and only if it preserves terminal objects and all pullbacks, and it is continuous if and only if it preserves terminal objects and all wide pullbacks. HINT: Exercises 3.2.48 and 3.2.50).

Dually, give similar characterisation of (finitely) cocontinuous functors, for instance as those that preserve (finite) coproducts and coequalisers. □

Exercise 3.4.24. Given a set X, show that the functor $[\_\to X]:\mathbf{Set}^{op} \to \mathbf{Set}$ from Exercise 3.4.16 is continuous. HINT: Use Exercise 3.4.23: relying on the explicit constructions of (co)products and (co)equalisers in **Set**, show that the functor maps any coproduct (disjoint union) of sets $\langle X_n \rangle_{n\in N}$ to a product of sets of functions $[X_n \to X]$, $n \in N$, and a coequaliser of functions $f,g:X_1 \to X_2$ to an equaliser of (pre-composition) functions $(f;\_),(g;\_):[X_2 \to X] \to [X_1 \to X]$.

You may also want to similarly check which of the examples of functors given above are (finitely) (co)continuous. □

Exercise 3.4.25. Consider a category **K** with a terminal object $1 \in |\mathbf{K}|$. Given any functor $\mathbf{F}:\mathbf{K} \to \mathbf{K}'$, check that **F** determines a functor $\mathbf{F}_{\downarrow 1}:\mathbf{K} \to \mathbf{K}'\downarrow\mathbf{F}(1)$ from **K** to the slice category of $\mathbf{K}'$-objects over $\mathbf{F}(1)$ (Definition 3.1.28), where for any object $A \in |\mathbf{K}|$, $\mathbf{F}_{\downarrow 1}(A) = \mathbf{F}(!_A)$, with $!_A:A \to 1$ being the unique morphism from A to 1, and $\mathbf{F}_{\downarrow 1}$ coincides with **F** on morphisms.

Suppose now that **K** has all pullbacks (so that it is finitely complete) and **F** preserves them (but we do not require **F** to preserve the terminal object, so it does not have to be finitely continuous). Show that $\mathbf{F}_{\downarrow 1}:\mathbf{K} \to \mathbf{K}'\downarrow\mathbf{F}(1)$ is finitely continuous. HINT: Recall Exercise 3.2.51. By the discussion there, since **F** preserves pullbacks, **F** maps products in **K**, which are pullback of morphisms to 1, to pullbacks in $\mathbf{K}'$ of morphisms to $\mathbf{F}(1)$ — and these are essentially products in $\mathbf{K}'\downarrow\mathbf{F}(1)$. Moreover, by the construction, $\mathbf{F}_{\downarrow 1}$ preserves the terminal object, and the conclusion follows by Exercise 3.4.23.

Similarly, show that if **K** has all wide pullbacks (so that it is complete) and **F** preserves them then $\mathbf{F}_{\downarrow 1}:\mathbf{K} \to \mathbf{K}'\downarrow\mathbf{F}(1)$ is continuous. □

Exercise 3.4.26. Recall the definition of the category $\mathbf{T}_{\Sigma,\Phi}$, the algebraic theory generated by a set Φ of equations over a signature Σ (cf. Exercise 3.1.15). Show that those functors from $\mathbf{T}^{op}_{\Sigma,\varnothing}$ to **Set** that preserve finite products (where products in $\mathbf{T}^{op}_{\Sigma,\Phi}$, that is coproducts in $\mathbf{T}_{\Sigma,\Phi}$, are given by concatenation of sequences of sort names, cf. Exercise 3.2.18, and products in **Set** are given by the Cartesian product) are in a bijective correspondence with Σ-algebras in $|\mathbf{Alg}(\Sigma)|$. Generalise this correspondence further to product-preserving functors from $\mathbf{T}^{op}_{\Sigma,\Phi}$ to **Set** and Σ-algebras in $Mod_\Sigma(\Phi)$. □

Definition 3.4.27 (Functor composition). The category **Cat** (the category of all categories) is defined as follows:

Objects of **Cat**: categories[5];
Morphisms of **Cat**: functors;
Composition in **Cat**: If $\mathbf{F}\colon \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G}\colon \mathbf{K2} \to \mathbf{K3}$ are functors, then $\mathbf{F};\mathbf{G}\colon \mathbf{K1} \to \mathbf{K3}$ is a functor defined as follows: $(\mathbf{F};\mathbf{G})_{Obj} = \mathbf{F}_{Obj};\mathbf{G}_{Obj}$ and $(\mathbf{F};\mathbf{G})_{A,B} = \mathbf{F}_{A,B};\mathbf{G}_{\mathbf{F}(A),\mathbf{F}(B)}$ for all $A, B \in \mathbf{K1}$. □

Example 3.4.28. In the following we will often use the functor $|\_|\colon \mathbf{Cat} \to \mathbf{Set}$[6] which for any category $\mathbf{K} \in |\mathbf{Cat}|$ yields the collection $|\mathbf{K}|$ of the objects of this category and for each functor $\mathbf{F}\colon \mathbf{K} \to \mathbf{K}'$ yields its object part $|\mathbf{F}| = \mathbf{F}_{Obj}\colon |\mathbf{K}| \to |\mathbf{K}'|$. □

Example 3.4.29. $\mathbf{Alg}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ is a functor, where:

- for any $\Sigma \in |\mathbf{AlgSig}|$, $\mathbf{Alg}(\Sigma)$ is the category of Σ-algebras; and
- for any morphism $\sigma\colon \Sigma \to \Sigma'$ in **AlgSig**, $\mathbf{Alg}(\sigma)$ is the reduct functor $\_|_\sigma\colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$. □

Exercise 3.4.30. Define a functor $\mathbf{Alg}^{der}\colon (\mathbf{AlgSig}^{der})^{op} \to \mathbf{Cat}$ so that $\mathbf{Alg}^{der}(\Sigma) = \mathbf{Alg}(\Sigma)$ for any signature $\Sigma \in |\mathbf{AlgSig}^{der}|$, and for any derived signature morphism δ, $\mathbf{Alg}^{der}(\delta)$ is the δ-reduct as sketched in Definition 1.5.16 and Exercise 1.5.17. □

Exercise 3.4.31. Define the category **Poset** (objects: partially-ordered sets; morphisms: order-preserving functions). Define the functor from **Poset** to **Cat** that maps a partially-ordered set to the corresponding (preorder) category (cf. Example 3.1.3) and an order-preserving function to the corresponding functor. □

Exercise 3.4.32. Characterise isomorphisms in **Cat**. Show that product categories are products in **Cat**. What are terminal objects, pullbacks and equalisers in **Cat**? Conclude that **Cat** is complete. HINT: Use constructions analogous to those in **Set**, as summarised in Exercise 3.2.53. □

Exercise 3.4.33. Prove that $\mathbf{Alg}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ (cf. Example 3.4.29) is continuous, that is, that it maps colimits in the category **AlgSig** of signatures to limits in the category **Cat** of all categories.

HINT: By Exercise 3.4.23 it is enough to show that **Alg** maps coproducts of signatures to products of the corresponding categories of algebras and coequalisers of signature morphisms to equalisers of the corresponding reduct functors.

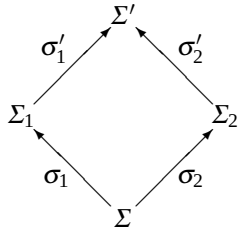
(*Coproducts*): Recall that by Exercise 3.2.16, a coproduct of signatures is in fact their disjoint union. Now, it is easy to see that an algebra over a disjoint union of a family of signatures may be identified with a tuple of algebras over the signatures in the family. Since a similar fact holds for homomorphisms, the rest of the proof in this case is straightforward (cf. Exercise 3.4.32). Notice that this argument covers the coproduct of the empty family of signatures as well.

[5] To be cautious about the set-theoretic foundations here, we should rather say: *small* categories.

[6] Again, we should restrict attention to small categories here. Alternatively, in place of **Set** we could use the category of all discrete categories, inheriting all of the foundational problems of **Cat**.

(*Coequalisers*): Recall (cf. Exercise 3.2.24) that a coequaliser of two signature morphisms $\sigma, \sigma' : \Sigma \to \Sigma'$ is the natural projection $p : \Sigma' \to (\Sigma'/\equiv)$, where $\equiv$ is the least equivalence relation on Σ' such that $\sigma(x) \equiv \sigma'(x)$ for all sort and operation names x in Σ (this is just a sketch of the construction). Notice now that $(\Sigma'/\equiv)$-algebras correspond exactly to those Σ'-algebras that have identical components $\sigma(x)$ and $\sigma'(x)$ for all sort and operation names x in Σ, or equivalently, to those algebras $A' \in |\mathbf{Alg}(\Sigma')|$ for which $A'|_\sigma = A'|_{\sigma'}$. Moreover, the correspondence is given by the functor $\_|_p : \mathbf{Alg}(\Sigma'/\equiv) \to \mathbf{Alg}(\Sigma')$. Since a similar fact holds for homomorphisms, it is straightforward now to prove that $\_|_p = \mathbf{Alg}(p)$ is an equaliser of $\_|_\sigma = \mathbf{Alg}(\sigma)$ and $\_|_{\sigma'} = \mathbf{Alg}(\sigma')$ (cf. Exercises 3.4.32 and 3.2.22). □

Exercise 3.4.34 (Amalgamation Lemma for algebras). Consider a pushout in the category **AlgSig** of signatures:



Conclude from Exercise 3.4.33 above that for any Σ_1-algebra A_1 and Σ_2-algebra A_2 such that $A_1|_{\sigma_1} = A_2|_{\sigma_2}$, there exists a unique Σ'-algebra A' such that $A'|_{\sigma'_1} = A_1$ and $A'|_{\sigma'_2} = A_2$.

Similarly, for any two homomorphisms $h_1 : A_{11} \to A_{12}$ in $\mathbf{Alg}(\Sigma_1)$ and $h_2 : A_{21} \to A_{22}$ in $\mathbf{Alg}(\Sigma_2)$ such that $h_1|_{\sigma_1} = h_2|_{\sigma_2}$, there exists a unique Σ'-homomorphism $h' : A'_1 \to A'_2$ such that $h'|_{\sigma'_1} = h_1$ and $h'|_{\sigma'_2} = h_2$. □

Example 3.4.35. Recall Example 3.2.35 of a simple pushout of algebraic signatures. Let $N \in |\mathbf{Alg}(\Sigma\mathrm{NAT})|$ be the standard model of natural numbers. Build $N_1 \in |\mathbf{Alg}(\Sigma\mathrm{NAT}_{\mathit{fib}})|$ by adding to N the interpretation of the operation *fib* as the standard Fibonacci function, and $N_2 \in |\mathbf{Alg}(\Sigma\mathrm{NAT}_{\mathit{mult}})|$ by adding to N the interpretation of the operation *mult* as multiplication. By construction we have $N_1|_{\Sigma\mathrm{NAT}} = N = N_2|_{\Sigma\mathrm{NAT}}$ and so N_1 and N_2 amalgamate to a unique algebra $N' \in |\mathbf{Alg}(\Sigma\mathrm{NAT}_{\mathit{fib},\mathit{mult}})|$ such that $N'|_{\Sigma\mathrm{NAT}_{\mathit{fib}}} = N_1$ and $N'|_{\Sigma\mathrm{NAT}_{\mathit{mult}}} = N_2$. Clearly, N' is the only expansion of N that defines *fib* as the Fibonacci function (as N_1 does) and *mult* as multiplication (as N_2 does). □

Exercise 3.4.36. Define initial objects and coproducts in **Cat**. (HINT: This is easy.) Try to define coequalisers and then pushouts in **Cat**. (HINT: This is difficult.) □

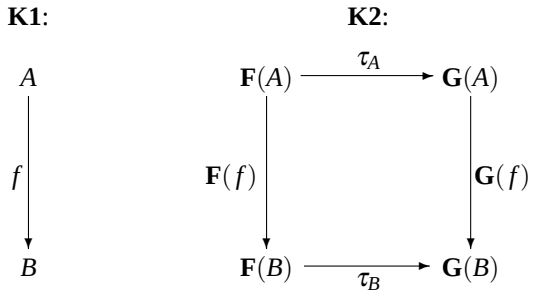
¡¡¡¡¡¡¡ c342.tex ======= ¿¿¿¿¿¿¿ 1.15

3.4.2 Natural transformations

Let $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G}:\mathbf{K1} \to \mathbf{K2}$ be two functors with common source and target categories.

A transformation from **F** to **G** should map the results of **F** to the results of **G**. This means, that it should consists of a family of morphisms in **K2**, one **K2**-morphism from $\mathbf{F}(A)$ to $\mathbf{G}(A)$ for each **K1**-object A. An extra requirement to impose is that this family should be compatible with the application of **F** and **G** to **K1**-morphisms, as formalised by the following definition:

Definition 3.4.37 (Natural transformation). A *natural transformation* from **F** to **G**, $\tau:\mathbf{F} \to \mathbf{G}$[7], is a family $\langle \tau_A:\mathbf{F}(A) \to \mathbf{G}(A)\rangle_{A\in|\mathbf{K1}|}$ of **K2**-morphisms such that for any $A,B \in |\mathbf{K1}|$ and **K1**-morphism $f:A \to B$ the following diagram commutes:



(this property is often referred to as the *naturality* of the family τ).

Furthermore, τ is a *natural isomorphism* if for all $A \in |\mathbf{K1}|$, τ_A is iso (in **K2**). □

Example 3.4.38. The identity transformation $id_{\mathbf{F}}:\mathbf{F} \to \mathbf{F}$, where $(id_{\mathbf{F}})_A = id_{\mathbf{F}(A)}$, is a natural isomorphism.

For any morphism $f:A \to B$ in a category **K2** and for any category **K1**, there is a constant natural transformation $\mathbf{c}_f:\mathbf{C}_A \to \mathbf{C}_B$ between the constant functors $\mathbf{C}_A,\mathbf{C}_B:\mathbf{K1} \to \mathbf{K2}$ (cf. Example 3.4.4) defined by $(\mathbf{c}_f)_o = f$ for all objects $o \in |\mathbf{K1}|$. □

Example 3.4.39. The family of singleton functions $sing\_set:\mathbf{Id_{Set}} \to \mathcal{P}$, where for any set X, $sing\_set_X:X \to \mathcal{P}(X)$ is defined by $sing\_set_X(a) = \{a\}$, is a natural transformation.

Let $(\_)^* = \mathbf{Seq};|\_|:\mathbf{Set} \to \mathbf{Set}$ be the functor given as the composition of $\mathbf{Seq}:\mathbf{Set} \to \mathbf{Mon}$ (Example 3.4.8) with the forgetful functor $|\_|:\mathbf{Mon} \to \mathbf{Set}$ mapping any monoid to its underlying carrier set. The family of singleton functions $sing\_seq:\mathbf{Id_{Set}} \to (\_)^*$, where for any set X, $sing\_seq_X:X \to X^*$ is defined by $sing\_seq_X(a) = a$ (*sing_seq* maps any element to the singleton sequence consisting of this element only) is a natural transformation. □

[7] Some authors would use a dotted or double arrow here, writing $\tau:\mathbf{F} \dot{\to} \mathbf{G}$ or $\tau:\mathbf{F} \Rightarrow \mathbf{G}$, respectively. We prefer to use the same symbol for all morphisms, and also for natural transformations, since they are morphisms in certain categories, see Definition 3.4.60 below.

Exercise 3.4.40. Consider the functor $(\_)^*:\mathbf{Set}\to\mathbf{Set}$ mapping any set X to the set X^* of sequences over X (cf. Example 3.4.39 above). Show that the following families of functions (indexed by sets $X\in|\mathbf{Set}|$) yield natural transformations from $(\_)^*$ to $(\_)^*$:

- for each $k\geq 0$, for $n\geq 0$ and $x_1,\ldots,x_n\in X$, $\mathit{stutter}^k_X(x_1\ldots x_n)=\underbrace{x_1\ldots x_1}_{k\text{ times}}\ldots\underbrace{x_n\ldots x_n}_{k\text{ times}}$;
- for each $k\geq 0$, for $n\geq 0$ and $x_1,\ldots,x_n\in X$, $\mathit{repeat}^k_X(x_1\ldots x_n)=\underbrace{x_1\ldots x_n\ldots x_1\ldots x_n}_{k\text{ times}}$;
- for $n\geq 0$ and $x_1,\ldots,x_n\in X$, $\mathit{reverse}_X(x_1\ldots x_n)=x_n\ldots x_1$;
- for $n\geq 0$ and $x_1,\ldots,x_{2n+1}\in X$, $\mathit{odds}_X(x_1x_2x_3\ldots x_{2n})=x_1x_3\ldots x_{2n-1}$ and $\mathit{odds}_X(x_1x_2x_3\ldots x_{2n+1})=x_1x_3\ldots x_{2n+1}$.

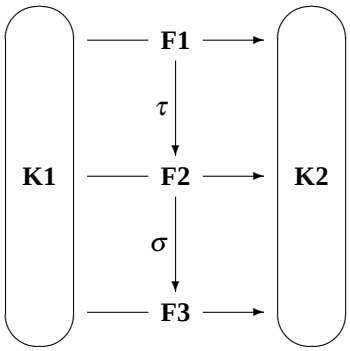
Check which of these functions also yield natural transformations from **Seq** to **Seq** (where $\mathbf{Seq}:\mathbf{Set}\to\mathbf{Mon}$, cf. Example 3.4.8).

The above examples indicate a close link between polymorphic functions as encountered in functional programming languages (like Standard ML [MTHM97] or Haskell [Pey03]) and natural transformations between functors representing polymorphic types. This property, often referred to as "parametric polymorphism" (as opposed to "ad hoc polymorphism") can be explored to derive some propeties of polymorphic functions directly from their types [Wad89]. □

Exercise 3.4.41. Recall (Exercise 3.4.26) the correspondence between product-preserving functors from $\mathbf{T}^{op}_{\Sigma,\Phi}$ to **Set** and Σ-algebras in $|\mathbf{Mod}(\Sigma,\Phi)|$. Show that this correspondence extends to morphisms: each Σ-homomorphism between algebras gives rise to a natural transformation between the corresponding functors, and vice versa, each natural transformation between such functors determines a homomorphism between the corresponding algebras. HINT: To prove that this yields a bijective correspondence, first use the naturality condition for product projections to show that for any natural transformation $\tau:\mathbf{F}\to\mathbf{G}$ between product-preserving functors $\mathbf{F},\mathbf{G}:\mathbf{T}^{op}_{\Sigma,\Phi}\to\mathbf{Set}$, any sequence $s_1\ldots s_n$ of sort names (an object in $\mathbf{T}_{\Sigma,\Phi}$) and any $\langle a_1,\ldots,a_n\rangle\in\mathbf{F}(s_1\ldots s_n)$, $\tau_{s_1\ldots s_n}(\langle a_1,\ldots,a_n\rangle)=\langle\tau_{s_1}(a_1),\ldots,\tau_{s_n}(a_n)\rangle$. □

Natural transformations have been introduced as morphisms between functors. The obvious thing to do next is to define composition of natural tranformations. Traditionally, two different composition operations for natural transformations are introduced: *vertical* and *horizontal* composition. The former is a straightforward composition of natural transformations between parallel functors. The latter is somewhat more involved; in a sense, it shows how natural transformations "accumulate" when functors are composed.

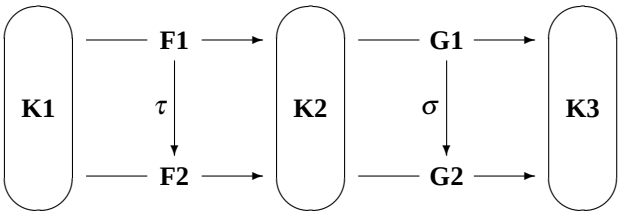
Definition 3.4.42 (Vertical composition). Let $\mathbf{F1},\mathbf{F2},\mathbf{F3}:\mathbf{K1}\to\mathbf{K2}$ be three functors with common source and target categories. Let $\tau:\mathbf{F1}\to\mathbf{F2}$ and $\sigma:\mathbf{F2}\to\mathbf{F3}$ be natural transformations:



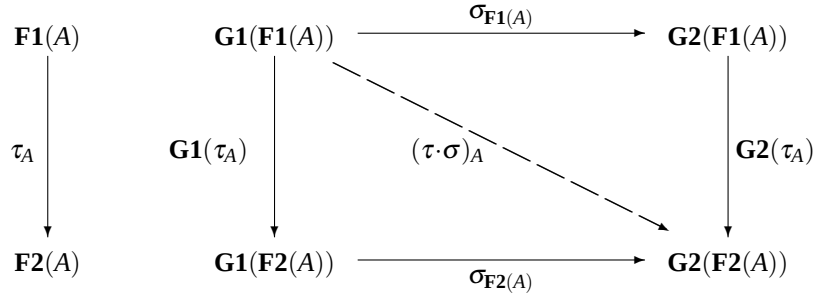
Then the *vertical composition* of τ and σ, $\tau;\sigma:\mathbf{F1}\to\mathbf{F3}$, is defined by $(\tau;\sigma)_A = \tau_A;\sigma_A$ (in **K2**) for all $A\in|\mathbf{K1}|$. □

Exercise 3.4.43. Prove that $\tau;\sigma$ is indeed a natural transformation. □

Definition 3.4.44 (Horizontal composition). Let $\mathbf{F1},\mathbf{F2}:\mathbf{K1}\to\mathbf{K2}$ and $\mathbf{G1},\mathbf{G2}:\mathbf{K2}\to\mathbf{K3}$ be two pairs of parallel functors. Let $\tau:\mathbf{F1}\to\mathbf{F2}$ and $\sigma:\mathbf{G1}\to\mathbf{G2}$ be natural transformations:

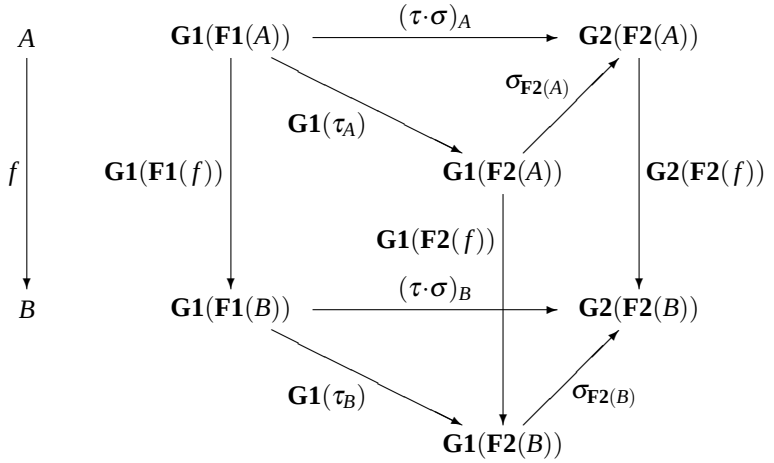


Then the *horizontal composition* of τ and σ, $\tau\cdot\sigma:\mathbf{F1};\mathbf{G1}\to\mathbf{F2};\mathbf{G2}$, is defined by $(\tau\cdot\sigma)_A = \mathbf{G1}(\tau_A);\sigma_{\mathbf{F2}(A)} = \sigma_{\mathbf{F1}(A)};\mathbf{G2}(\tau_A)$ (in **K3**) for all $A\in|\mathbf{K1}|$:



□

Exercise 3.4.45. Prove that the above diagram commutes, and so $(\tau\cdot\sigma)_A$ is well-defined. Then prove that $\tau\cdot\sigma$ is indeed a natural transformation. HINT:



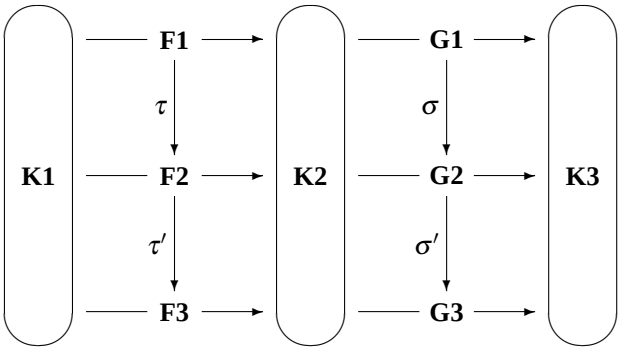
□

Definition 3.4.46 (Multiplication by a functor). A special case of the horizontal composition of natural transformations is the *multiplication* of a natural transformation by a functor. Under the assumptions of Definition 3.4.44, we define:

- $\tau\cdot\mathbf{G1} = \tau\cdot id_{\mathbf{G1}}\colon \mathbf{F1};\mathbf{G1} \to \mathbf{F2};\mathbf{G1}$, or more explicitly: $(\tau\cdot\mathbf{G1})_A = \mathbf{G1}(\tau_A)$ for $A \in |\mathbf{K1}|$;
- $\mathbf{F1}\cdot\sigma = id_{\mathbf{F1}}\cdot\sigma\colon \mathbf{F1};\mathbf{G1} \to \mathbf{F1};\mathbf{G2}$, or more explicitly: $(\mathbf{F1}\cdot\sigma)_A = \sigma_{\mathbf{F1}(A)}$ for $A \in |\mathbf{K1}|$. □

Exercise 3.4.47. Show that $\tau\cdot\sigma = (\tau\cdot\mathbf{G1});(\mathbf{F2}\cdot\sigma) = (\mathbf{F1}\cdot\sigma);(\tau\cdot\mathbf{G2})$. □

Exercise 3.4.48 (Interchange law). Consider any categories **K1**, **K2**, **K3**, functors $\mathbf{F1},\mathbf{F2},\mathbf{F3}\colon \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G1},\mathbf{G2},\mathbf{G3}\colon \mathbf{K2} \to \mathbf{K3}$, and natural transformations $\tau\colon \mathbf{F1} \to \mathbf{F2}$, $\tau'\colon \mathbf{F2} \to \mathbf{F3}$, $\sigma\colon \mathbf{G1} \to \mathbf{G2}$, and $\sigma'\colon \mathbf{G2} \to \mathbf{G3}$:



Show that $(\tau;\tau')\cdot(\sigma;\sigma') = (\tau\cdot\sigma);(\tau'\cdot\sigma')$. □

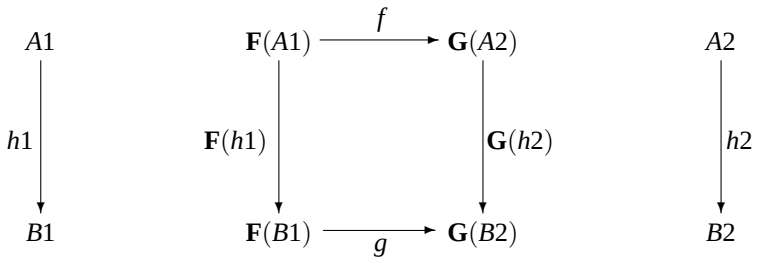
3.4.3 Constructing categories, revisited

3.4.3.1 Comma categories

Definition 3.4.49 (Comma category). Let $\mathbf{F}:\mathbf{K1}\to\mathbf{K}$ and $\mathbf{G}:\mathbf{K2}\to\mathbf{K}$ be two functors with a common target category. The *comma category* $(\mathbf{F},\mathbf{G})$ is defined by:

Objects of $(\mathbf{F},\mathbf{G})$: triples $\langle A1, f, A2\rangle$, where $A1\in|\mathbf{K1}|$, $A2\in|\mathbf{K2}|$ and $f:\mathbf{F}(A1)\to\mathbf{G}(A2)$ is a morphism in $\mathbf{K}$;

Morphisms of $(\mathbf{F},\mathbf{G})$: a morphism from $\langle A1, f, A2\rangle$ to $\langle B1, g, B2\rangle$ is a pair $\langle h1, h2\rangle$ of morphisms where $h1:A1\to B1$ (in **K1**) and $h2:A2\to B2$ (in **K2**) such that (the middle part of) the following diagram commutes:



Composition in $(\mathbf{F},\mathbf{G})$: $\langle h1,h2\rangle;\langle h1',h2'\rangle=\langle h1;h1',h2;h2'\rangle$. □

Exercise 3.4.50. Construct the category $\mathbf{K}^{\to}$ of **K**-morphisms and the category $\mathbf{K}{\downarrow}A$ of **K**-objects over $A\in|\mathbf{K}|$ as comma categories (cf. Definitions 3.1.27 and 3.1.28). HINT: Consider categories $(\mathbf{Id_K},\mathbf{Id_K})$ and $(\mathbf{Id}_K,\mathbf{C}^{\mathbf{1}}_A)$, where $\mathbf{Id_K}$ is the identity functor on **K** and $\mathbf{C}^{\mathbf{1}}_A:\mathbf{1}\to\mathbf{K}$ is a constant functor from the terminal category **1**. □

Example 3.4.51. Another way of presenting the category **Graph** is as the comma category $(\mathbf{Id_{Set}},\mathbf{CP})$, where $\mathbf{CP}:\mathbf{Set}\to\mathbf{Set}$ is the Cartesian product functor defined by $\mathbf{CP}(X)=X\times X$ and $\mathbf{CP}(f:X\to Y)\langle x1,x2\rangle=\langle f(x1),f(x2)\rangle$.

To see this, write an object in $|(\mathbf{Id_{Set}},\mathbf{CP})|$ as $\langle E,\langle source:E\to N, target:E\to N\rangle, N\rangle$. □

Exercise 3.4.52. Another way to present the category of signatures **AlgSig** is as the comma category $(\mathbf{Id_{Set}},(\_)^+)$, where $(\_)^+:\mathbf{Set}\to\mathbf{Set}$ is the functor which for any set $X\in|\mathbf{Set}|$ yields the set X^+ of all finite non-empty sequences of elements from X.

First, complete the definition of the functor $(\_)^+$. Then, notice that $X^+=X^*\times X$ and hence an object in $|(\mathbf{Id_{Set}},(\_)^+)|$ may be written as $\langle\Omega,\langle arity:\Omega\to S^*, sort:\Omega\to S\rangle, S\rangle$. Indicate now why the category defined is almost, but not quite, the same as the category **AlgSig** of signatures (cf. Exercise 3.4.75 below). □

Exercise 3.4.53. Prove that if **K1** and **K2** are (finitely) complete categories, $\mathbf{F}:\mathbf{K1}\to\mathbf{K}$ is a functor, and $\mathbf{G}:\mathbf{K2}\to\mathbf{K}$ is a (finitely) continuous functor, then the comma category $(\mathbf{F},\mathbf{G})$ is (finitely) complete. Moreover, the obvious projections from $(\mathbf{F},\mathbf{G})$ to **K1** and **K2**, respectively, are (finitely) continuous. HINT: To construct a limit

of a diagram in $(\mathbf{F},\mathbf{G})$, start by building limits of the projections of the diagram to **K1** and **K2**, respectively, and then use the continuity property of **G** to complete the construction of the limit object in $(\mathbf{F},\mathbf{G})$. If the notation in the proof gets too heavy, use Exercise 3.2.49 and spell the details out for the construction of products and equalisers.

Check that this construction of limits in $(\mathbf{F},\mathbf{G})$ works for diagrams of any given shape: if **K1** and **K2** have limits of diagrams of a given shape, and **G** preserves them, then $(\mathbf{F},\mathbf{G})$ has limits of diagrams of this shape, and the projection functors preserve them.

State and prove the analogous facts about cocompleteness of $(\mathbf{F},\mathbf{G})$. HINT: Clearly, appropriate colimits must exist in **K1** and **K2**, but unlike with limits, it is **F** that must preserve them. □

Exercise 3.4.54. Use Exercises 3.4.50 and 3.4.53 to show that if **K** is a (finitely) complete category then so is the category $\mathbf{K}^{\rightarrow}$ of morphisms in **K**.

Then, without looking at Exercise 3.2.51, use Exercises 3.4.50 and 3.4.53 to prove that if a category **K** has limits of all (finite) non-empty connected diagrams then so does the slice category $\mathbf{K}{\downarrow}A$ of its objects over $A \in |\mathbf{K}|$, and that the obvious forgetful functor from $\mathbf{K}{\downarrow}A$ to **K** preserves these limits. Notice though that this does not generalise to arbitrary (finite) limits that exist in $\mathbf{K}{\downarrow}A$ if **K** is (finitely) complete by Exercise 3.2.51.

Check that your proof shows a stronger fact: without assuming the existence of any limits in **K**, the forgetful functor from $\mathbf{K}{\downarrow}A$ to **K** *creates* limits of all non-empty connected diagrams, that is: for any such diagram $D_{\downarrow A}$ in $\mathbf{K}{\downarrow}A$, if its projection D to **K** has a limit in **K** then there is a unique cocone on $D_{\downarrow A}$ in $\mathbf{K}{\downarrow}A$ that projects to this limit, and this cocone is a limit of $D_{\downarrow A}$ in $\mathbf{K}{\downarrow}A$. □

Exercise 3.4.55. Show that if **K** has all pullbacks and a terminal object (so, it is finitely complete) and a functor $\mathbf{F}\colon\mathbf{K}\to\mathbf{K}'$ preserves pullbacks, then **F** also preserves the limits of all finite non-empty connected diagrams. HINT: Put together Exercises 3.4.25 and 3.4.54.

Similarly, show that if **K** has all wide pullbacks and a terminal object (so, it is complete) and a functor $\mathbf{F}\colon\mathbf{K}\to\mathbf{K}'$ preserves wide pullbacks, then **F** also preserves the limits of all non-empty connected diagrams. □

3.4.3.2 Indexed categories

We frequently need to deal not just with a single category, but rather with a family of categories, "parameterised" by a certain collection of indices. The categories of S-sorted sets (one for each set S) and the categories of Σ-algebras (one for each signature Σ) are typical examples. A crucial property here is that all the categories in such a family are defined in a uniform way, and consequently any change of an index induces a smooth translation between the corresponding component categories. In typical examples, the translation goes in the opposite direction than the change of index, which leads to the following definition:

Definition 3.4.56 (Indexed category). An *indexed category* (over an *index category* **Ind**) is a functor $\mathbf{C}:\mathbf{Ind}^{op}\to\mathbf{Cat}$. □

Example 3.4.57. $\mathbf{Alg}:\mathbf{AlgSig}^{op}\to\mathbf{Cat}$ is an indexed category (cf. Example 3.4.29). □

Definition 3.4.58 (Grothendieck construction). Every indexed category $\mathbf{C}:\mathbf{Ind}^{op}\to\mathbf{Cat}$ gives rise to a *flattened* category $\mathbf{Flat}(\mathbf{C})$ defined as follows:

Objects of $\mathbf{Flat}(\mathbf{C})$: pairs $\langle i,A\rangle$ for all $i\in|\mathbf{Ind}|$ and $A\in|\mathbf{C}(i)|$;
Morphisms of $\mathbf{Flat}(\mathbf{C})$: a morphism from $\langle i,A\rangle$ to $\langle j,B\rangle$ is a pair $\langle\sigma,f\rangle:\langle i,A\rangle\to\langle j,B\rangle$, where $\sigma:i\to j$ is an **Ind**-morphism and $f:A\to\mathbf{C}(\sigma)(B)$ is a $\mathbf{C}(i)$-morphism;
Composition in $\mathbf{Flat}(\mathbf{C})$: $\langle\sigma,f\rangle;\langle\sigma',f'\rangle=\langle\sigma;\sigma',f;\mathbf{C}(\sigma)(f')\rangle$. □

Exercise 3.4.59. Show that if **Ind** is complete, $\mathbf{C}(i)$ are complete for all $i\in|\mathbf{Ind}|$, and $\mathbf{C}(\sigma)$ are continuous for all $\sigma\in\mathbf{Ind}$, then $\mathbf{Flat}(\mathbf{C})$ is complete.

HINT: Given a diagram in the flattened category $\mathbf{Flat}(\mathbf{C})$, first consider its obvious projection on the index category **Ind**. Since **Ind** is complete, this has a limit $l\in|\mathbf{Ind}|$. Using the functors assigned by **C** to the projection morphism of the limit, "translate" all the nodes and edges of the diagram to the category $\mathbf{C}(l)$, thus obtaining a diagram in $\mathbf{C}(l)$. Since $\mathbf{C}(l)$ is complete, it has a limit. Check that the projection morphisms of the limit of the diagram constructed in **Ind** when paired with the corresponding projection morphisms of the limit of the diagram in $\mathbf{C}(l)$ form the limit of the original diagram in $\mathbf{Flat}(\mathbf{C})$.

To make the construction manageable, consider only products and equalisers: this is sufficient by Exercise 3.2.49. □

3.4.3.3 Functor categories

Definition 3.4.60 (Functor category). Let **K1** and **K2** be categories[8]. The *functor category* [**K1**→**K2**] is defined by:

Objects of [**K1**→**K2**]: functors from **K1** to **K2**;
Morphisms of [**K1**→**K2**]: natural transformations;
Composition in [**K1**→**K2**]: vertical composition. □

Exercise 3.4.61. Define the category $\mathbf{Set}^S$ of S-sorted sets as a functor category. □

Exercise 3.4.62. For any category **K**, define its morphism category $\mathbf{K}^{\to}$ as the category of functors [**2**→**K**]. □

Exercise 3.4.63. Let **K1** and **K2** be categories. Show that if **K2** is (finitely) complete then so is the functor category [**K1**→**K2**]. State and show the dual fact as well.
HINT: The limit of any diagram in [**K1**→**K2**] may be constructed "pointwise", for

[8] To be cautious about set-theoretic foundations, one may want to assume that **K1** is small.

each object in $|\mathbf{K1}|$ separately. More precisely, using Exercise 3.2.49 to simplify the notational burden: consider any family of functors $\langle \mathbf{F}_n : \mathbf{K1} \to \mathbf{K2} \rangle_{n \in N}$. For each $X \in |\mathbf{K1}|$, let $\mathbf{Q}(X) \in |\mathbf{K2}|$ with projections $(\pi_n)_X : \mathbf{Q}(X) \to \mathbf{F}_n(X)$, $n \in N$, be a product of $\langle \mathbf{F}_n(X) \rangle_{n \in N}$ in $\mathbf{K2}$. Check that there is a unique way to extend $\mathbf{Q}$ to a functor $\mathbf{Q} : \mathbf{K1} \to \mathbf{K2}$ so that all $\pi_n : \mathbf{Q} \to \mathbf{F}_n$, $n \in N$, become natural transformations. Show that $\mathbf{Q}$ with projections $\langle \pi_n \rangle_{n \in N}$ is a product of $\langle \mathbf{F}_n : \mathbf{K1} \to \mathbf{K2} \rangle_{n \in N}$ in $[\mathbf{K1} \to \mathbf{K2}]$. Then proceed similarly for equalisers: consider functors $\mathbf{F}, \mathbf{F}' : \mathbf{K1} \to \mathbf{K2}$ and natural transformations $\tau_1, \tau_2 : \mathbf{F} \to \mathbf{F}'$. For each $X \in |\mathbf{K1}|$, let $\tau_X : \mathbf{Q}(X) \to \mathbf{F}(X)$ be an equaliser of $(\tau_1)_X, (\tau_2)_X : \mathbf{F}(X) \to \mathbf{F}'(X)$ in $\mathbf{K2}$. This yields a unique functor $\mathbf{Q} : \mathbf{K1} \to \mathbf{K2}$ such that $\tau : \mathbf{Q} \to \mathbf{F}$ is a natural transformation, which is an equaliser of τ_1, τ_2 in $[\mathbf{K1} \to \mathbf{K2}]$. □

Exercise 3.4.64. Let $\mathbf{K1}$, $\mathbf{K1}'$ and $\mathbf{K2}$ be categories. Show how any functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K1}'$ induces a functor $(\mathbf{F};\_) : [\mathbf{K1}' \to \mathbf{K2}] \to [\mathbf{K1} \to \mathbf{K2}]$. Relying on the construction outlined in Exercise 3.4.63 and assuming that $\mathbf{K2}$ is (finitely) complete, show that this functor is (finitely) continuous.

Prove also that this yields a functor $[\_ \to \mathbf{K2}] : \mathbf{Cat}^{op} \to \mathbf{Cat}$[9] (cf. Exercise 3.4.16). □

Exercise 3.4.65. For any category $\mathbf{K}$, define a category $\mathbf{Funct}(\mathbf{K})$ of *functors into* $\mathbf{K}$ as follows:

Objects of $\mathbf{Funct}(\mathbf{K})$: functors $\mathbf{F} : \mathbf{K}' \to \mathbf{K}$ into $\mathbf{K}$;
Morphisms of $\mathbf{Funct}(\mathbf{K})$: a morphism from $\mathbf{F} : \mathbf{K1} \to \mathbf{K}$ to $\mathbf{G} : \mathbf{K2} \to \mathbf{K}$ is a pair $\langle \Phi, \rho \rangle$, where $\Phi : \mathbf{K1} \to \mathbf{K2}$ is a functor and $\rho : \mathbf{F} \to \Phi;\mathbf{G}$ is a natural transformation (between functors from $\mathbf{K1}$ to $\mathbf{K}$);
Composition in $\mathbf{Funct}(\mathbf{K})$: $\langle \Phi, \rho \rangle ; \langle \Phi', \rho' \rangle = \langle \Phi;\Phi', \rho;(\Phi \cdot \rho') \rangle$.

Show how the category $\mathbf{Funct}(\mathbf{K})$ arises by the flattening construction of Definition 3.4.58 for the functor $[\_ \to \mathbf{K}]$ as defined in the previous exercise.[10] □

Exercise 3.4.66. Show that if $\mathbf{K}$ is a (finitely) complete category then the category $\mathbf{Funct}(\mathbf{K})$ of functors into $\mathbf{K}$ is (finitely) complete as well. HINT: You may construct the limits in $\mathbf{Funct}(\mathbf{K})$ directly, perhaps using Exercise 3.2.49. Alternatively, rely on the construction of $\mathbf{Funct}(\mathbf{K})$ by flattening (Definition 3.4.58) for the functor $[\_ \to \mathbf{K}] : \mathbf{Cat}^{op} \to \mathbf{Cat}$ and on Exercise 3.4.59; recall that $\mathbf{Cat}$ is complete by Exercise 3.4.32, for any category $\mathbf{K1}$, $[\mathbf{K1} \to \mathbf{K}]$ is (finitely) complete by Exercise 3.4.63, and for every functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$, $(\mathbf{F};\_) : [\mathbf{K2} \to \mathbf{K}] \to [\mathbf{K1} \to \mathbf{K}]$ is (finitely) continuous by Exercise 3.4.64. □

Exercise 3.4.67. Show that if a category $\mathbf{K1}$ has a factorisation system (cf. Section 3.3) than for any category $\mathbf{K2}$, the functor category $[\mathbf{K2} \to \mathbf{K1}]$ has a factorisation system as well.

HINT: Let $\langle \mathbf{E1}, \mathbf{M1} \rangle$ be a factorisation system for $\mathbf{K1}$. Define $\mathbf{E} = \{\varepsilon \in [\mathbf{K2} \to \mathbf{K1}] \mid \varepsilon_A \in \mathbf{E1} \text{ for } a \in |\mathbf{K2}|\}$ and $\mathbf{M} = \{\eta \in [\mathbf{K2} \to \mathbf{K1}] \mid \eta_A \in \mathbf{M1} \text{ for } a \in |\mathbf{K2}|\}$. Now,

[9] Assuming that $\mathbf{K2}$ is small would help to resolve potential foundational problems here.

[10] So, for foundational reasons, one may prefer to keep all categories small around here as well.

to construct an $\langle \mathbf{E}, \mathbf{M} \rangle$-factorisation of a natural transformation $\tau: \mathbf{F} \to \mathbf{G}$ between functors $\mathbf{F}, \mathbf{G}: \mathbf{K2} \to \mathbf{K1}$, first for each object $a \in |\mathbf{K2}|$ obtain an $\langle \mathbf{E1}, \mathbf{M1} \rangle$-factorisation of τ_A, say $\tau_A = \varepsilon_A; \eta_A$ with $\varepsilon_A \in \mathbf{E1}$ and $\eta_A \in \mathbf{M1}$, and $\varepsilon_A: \mathbf{F}(A) \to \mathbf{H}(A)$, $\eta_A: \mathbf{H}(A) \to \mathbf{G}(A)$ for some $\mathbf{H}(A) \in |\mathbf{K1}|$. Then use the diagonal fill-in lemma (Lemma 3.3.4) to extend the mapping $\mathbf{H}: |\mathbf{K2}| \to |\mathbf{K1}|$ to a functor $\mathbf{H}: \mathbf{K2} \to \mathbf{K1}$ such that $\varepsilon: \mathbf{F} \to \mathbf{H}$ and $\eta: \mathbf{H} \to \mathbf{G}$ are natural transformations. □

3.4.3.4 Equivalence of categories

Definition 3.4.68 (Isomorphic categories). Two categories **K1** and **K2** are *isomorphic* if there are functors $\mathbf{F}: \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{F}^{-1}: \mathbf{K2} \to \mathbf{K1}$ such that $\mathbf{F}; \mathbf{F}^{-1} = \mathbf{Id}_{\mathbf{K1}}$ and $\mathbf{F}^{-1}; \mathbf{F} = \mathbf{Id}_{\mathbf{K2}}$. □

In other words, we say that two categories are isomorphic if they are isomorphic as objects of **Cat**. As with isomorphic objects of other kinds, we will view isomorphic categories as abstractly the same. It turns out, however, that in this case there is a coarser relation which allows us to identify categories which have all the same categorical properties, even though they may not be isomorphic.

Definition 3.4.69 (Equivalent categories). **K1** and **K2** are *equivalent* if there are functors $\mathbf{F}: \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G}: \mathbf{K2} \to \mathbf{K1}$ and natural isomorphisms $\tau: \mathbf{Id}_{\mathbf{K1}} \to \mathbf{F}; \mathbf{G}$ and $\sigma: \mathbf{G}; \mathbf{F} \to \mathbf{Id}_{\mathbf{K2}}$. □

To characterise equivalent categories, we need one more concept:

Definition 3.4.70 (Skeletal category). A category **K** is *skeletal* iff any two isomorphic **K**-objects are identical. A *skeleton of* **K** is any maximal skeletal subcategory of **K**. □

Exercise 3.4.71. Prove that two categories are equivalent iff they have isomorphic skeletons. □

Thus, intuitively, two categories are equivalent if and only if they differ only in the number of isomorphic copies of corresponding objects.

Example 3.4.72. The category **FinSet** of all finite sets is equivalent to its full subcategory of all natural numbers, where any natural number n is defined as the set $\{0, \ldots, n-1\}$ of all natural numbers smaller than n. In fact, the latter is a skeleton of **FinSet**. Similarly, the category **Set** of all sets is equivalent to its full subcategory of all ordinals. □

Exercise 3.4.73. Show that for any signature Σ and set Φ of Σ-equations, the full subcategory of $\mathbf{T}_\Sigma/\Phi$ given by the finite sets of variables is equivalent to the category $\mathbf{T}_{\Sigma,\Phi}$ (cf. Exercises 3.1.14 and 3.1.15). □

Exercise 3.4.74. Let **K1** and **K2** be equivalent categories. Show that if **K1** is (finitely) (co)complete then so is **K2**. □

Exercise 3.4.75. Recall Exercise 3.4.52. As indicated there, categories **AlgSig** and $(\mathbf{Id}_{\mathbf{Set}}, (\_)^+)$ are not isomorphic. Show that they are equivalent. Then, using Exercises 3.4.74 and 3.4.53, conclude from this that **AlgSig** is complete and cocomplete. □

3.5 Adjoints

Recall Facts 1.4.4 and 1.4.10:

Fact 1.4.4. *For any Σ-algebra A and S-sorted function $v: X \to |A|$ there is exactly one Σ-homomorphism $v^\#: T_\Sigma(X) \to A$ which extends v, i.e. such that $v^\#_s(\iota_X(x)) = v_s(x)$ for all $s \in S$, $x \in X_s$, where $\iota_X: X \to |T_\Sigma(X)|$ is the embedding that maps each variable in X to the corresponding term.* □

Fact 1.4.10. *This property defines $T_\Sigma(X)$ up to isomorphism: if B is a Σ-algebra and $\eta: X \to |B|$ is an S-sorted function such that for any Σ-algebra A and S-sorted function $v: X \to |A|$ there is a unique Σ-homomorphism $v^\$: B \to A$ such that $\eta; |v^\$| = v$ then B is isomorphic to $T_\Sigma(X)$.* □

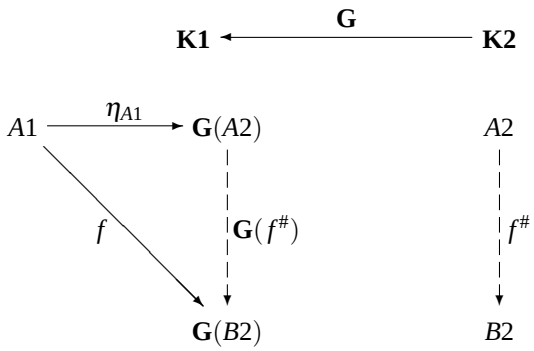
The construction of the algebra of Σ-terms is one example of an *adjoint functor* (it is *left adjoint* to the functor $|\_|: \mathbf{Alg}(\Sigma) \to \mathbf{Set}^{sorts(\Sigma)}$). The general concept of an adjoint functor, to which this section is devoted, has many other important instances. In fact, [Gog91b] goes so far as to say:

> *Any canonical construction from widgets to whatsits is an adjoint of another functor, from whatsits to widgets.*

3.5.1 Free objects

Let **K1** and **K2** be categories, $\mathbf{G}: \mathbf{K2} \to \mathbf{K1}$ be a functor, and $A1$ be an object of **K1**.

Definition 3.5.1 (Free object). A *free object over $A1$ w.r.t.* $\mathbf{G}$ is a **K2**-object $A2$ together with a **K1**-morphism $\eta_{A1}: A1 \to \mathbf{G}(A2)$ such that for any **K2**-object $B2$ and **K1**-morphism $f: A1 \to \mathbf{G}(B2)$ there is a unique **K2**-morphism $f^\#: A2 \to B2$ such that $\eta_{A1}; \mathbf{G}(f^\#) = f$.



η_{A1} is called the *unit morphism*. □

Example 3.5.2. Let $\Sigma = \langle S, \Omega \rangle$ be an arbitrary signature. Consider the forgetful functor $|\_|: \mathbf{Alg}(\Sigma) \to \mathbf{Set}^S$. Fact 1.4.4 asserts that for any S-sorted set X, the term algebra $T_\Sigma(X)$ with the inclusion $\eta_X: X \hookrightarrow |T_\Sigma(X)|$ is a free object over X w.r.t. $|\_|$. □

Exercise 3.5.3. Define free monoids and the path categories $\mathbf{Path}(G)$ as free objects w.r.t. some obvious functors. Then, look around at the areas of mathematics with which you are familiar for more examples. For instance, check that free groups and discrete topologies, (ideal) completion of partial orders, of ordered algebras, etc. may be defined as free objects w.r.t. some straightforward functors. □

Exercise 3.5.4. Prove that any free object over $A1$ w.r.t. $\mathbf{G}$ is an initial object in the comma category $(\mathbf{C}_{A1}, \mathbf{G})$, where $\mathbf{C}_{A1}: \mathbf{1} \to \mathbf{K1}$ is the constant functor. Conclude that a free object over $A1$ w.r.t. $\mathbf{G}$ is unique up to isomorphism. □

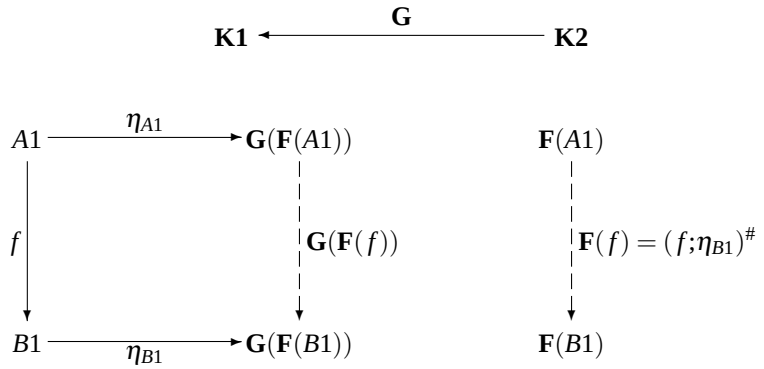
Exercise 3.5.5. Prove that if $A2 \in |\mathbf{K2}|$ is a free object over $A1 \in |\mathbf{K1}|$ w.r.t. $\mathbf{G}: \mathbf{K2} \to \mathbf{K1}$, then for any $B2 \in |\mathbf{K2}|$, $\#: \mathbf{K1}(A1, \mathbf{G}(B2)) \to \mathbf{K2}(A2, B2)$ is a bijection.

Check that one consequence of this is that two morphisms $g, h: A2 \to B2$ coincide (in $\mathbf{K2}$) whenever $\eta_{A1}; \mathbf{G}(g) = \eta_{A1}; \mathbf{G}(h)$ in $\mathbf{K1}$. □

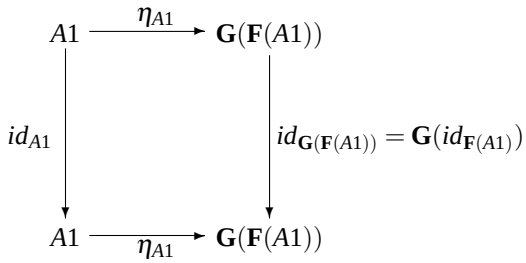
3.5.2 Left adjoints

Let $\mathbf{K1}$ and $\mathbf{K2}$ be categories and $\mathbf{G}: \mathbf{K2} \to \mathbf{K1}$ be a functor. So far we have considered free objects w.r.t. $\mathbf{G}$ one by one, without relating them with each other. One crucial property is that the construction of free objects, if they exist, is functorial.

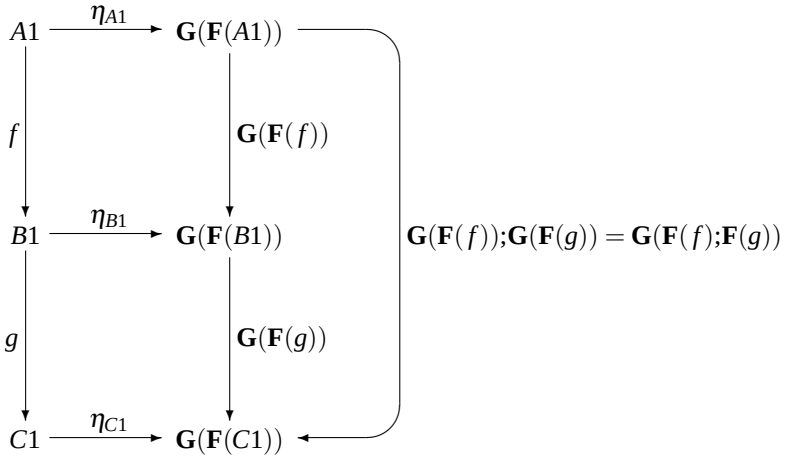
Proposition 3.5.6. *If for any $A1 \in |\mathbf{K1}|$ there is a free object over $A1$ w.r.t. $\mathbf{G}$, say $F(A1) \in |\mathbf{K2}|$ with unit morphism $\eta_{A1}: A1 \to \mathbf{G}(F(A1))$ (in $\mathbf{K1}$), then $A1 \mapsto F(A1)$ and $f \in \mathbf{K1}(A1, B1) \mapsto (f; \eta_{B1})^\# \in \mathbf{K2}(F(A1), F(B1))$ determine a functor $\mathbf{F}: \mathbf{K1} \to \mathbf{K2}$.*



Proof. **F** *preserves identities*: $\mathbf{F}(id_{A1}) = (id_{A1};\eta_{A1})^{\#} = id_{\mathbf{F}(A1)}$ follows from the fact that the following diagram commutes:



F *preserves composition*: Since the following diagram commutes:



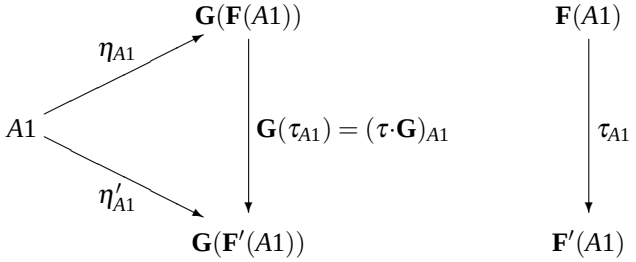
it follows that $\mathbf{F}(f;g) = (f;g;\eta_{C1})^{\#} = \mathbf{F}(f);\mathbf{F}(g)$. □

Exercise 3.5.7. Prove that $\eta : \mathbf{Id_{K1}} \to \mathbf{F};\mathbf{G}$ in Proposition 3.5.6 is a natural transformation. □

Definition 3.5.8 (Left adjoint). Let $\mathbf{F}:\mathbf{K1}\to\mathbf{K2}$ and $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$ be functors and $\eta:\mathbf{Id_{K1}}\to\mathbf{F};\mathbf{G}$ be a natural transformation. $\mathbf{F}$ is *left adjoint to* $\mathbf{G}$ *with unit* η if for any $A1\in|\mathbf{K1}|$, $\mathbf{F}(A1)$ with unit morphism $\eta_{A1}:A1\to\mathbf{G}(\mathbf{F}(A1))$ is a free object over $A1$ w.r.t. $\mathbf{G}$. □

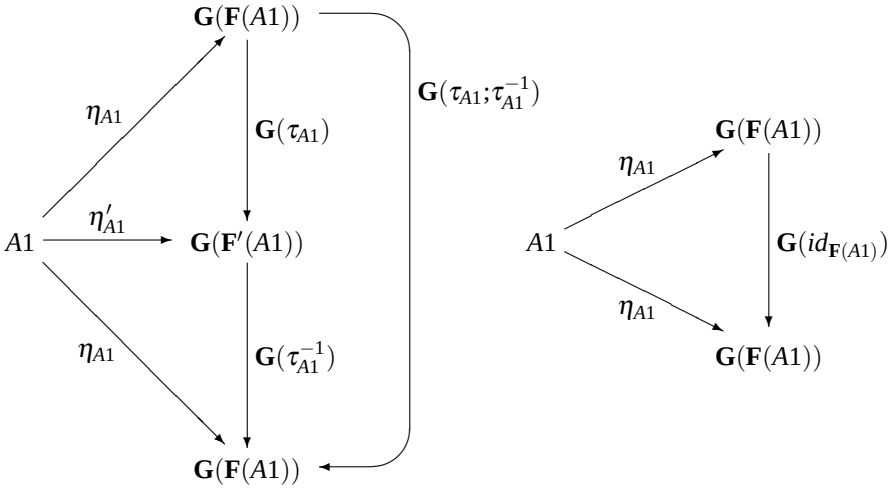
Before we give any examples, let us prove a very important property of left adjoints.

Proposition 3.5.9. *A left adjoint to* $\mathbf{G}$ *is unique up to (natural) isomorphism: if* $\mathbf{F}$ *and* $\mathbf{F}'$ *are left adjoints of* $\mathbf{G}$ *with units* η *and* η' *respectively, then there is a natural isomorphism* $\tau:\mathbf{F}\to\mathbf{F}'$ *such that* $\eta;(\tau\cdot\mathbf{G})=\eta'$.



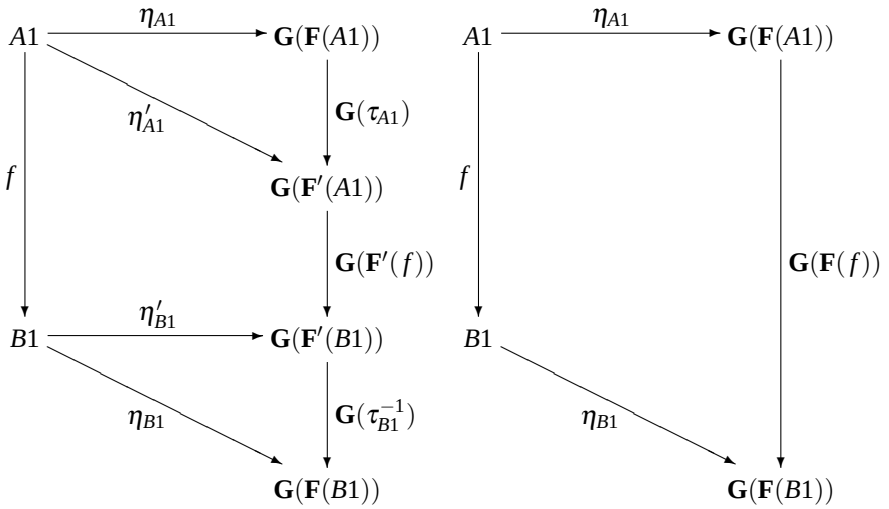
Proof. First notice that for any $f\in\mathbf{K1}(A1,B1)$, $\mathbf{F}(f)=(f;\eta_{B1})^{\#}$ and $\mathbf{F}'(f)=(f;\eta'_{B1})^{\#'}$.

Then, for $A1\in|\mathbf{K1}|$, define $\tau_{A1}=(\eta'_{A1})^{\#}$ and $\tau_{A1}^{-1}=(\eta_{A1})^{\#'}$. Then $\tau_{A1};\tau_{A1}^{-1}=id_{\mathbf{F}(A1)}$ since the following diagrams commute:



and $\tau_{A1}^{-1};\tau_{A1}=id_{\mathbf{F}'(A1)}$ by a similar argument.

Finally, for $f:A1\to B1$ (in **K1**), the following diagrams commute:



Thus, $\mathbf{F}(f) = (f;\eta_{B1})^{\#} = \tau_{A1};\mathbf{F}'(f);\tau_{B1}^{-1}$. This proves that $\mathbf{F}(f);\tau_{B1} = \tau_{A1};\mathbf{F}'(f)$, and hence that $\tau:\mathbf{F} \to \mathbf{F}'$ is natural. □

Example 3.5.10. For any signature $\Sigma = \langle S, \Omega \rangle$, the functor $T_\Sigma:\mathbf{Set}^S \to \mathbf{Alg}(\Sigma)$ is left adjoint to the forgetful functor $|\_|:\mathbf{Alg}(\Sigma) \to \mathbf{Set}^S$ (cf. Examples 3.4.11 and 3.4.9).

The functor $\mathbf{Seq}:\mathbf{Set} \to \mathbf{Mon}$ is left adjoint to the forgetful functor $|\_|:\mathbf{Mon} \to \mathbf{Set}$ which takes a monoid to its underlying set of elements. The unit is $\mathit{sing\_seq}:\mathbf{Id}_{\mathbf{Set}} \to \mathbf{Seq};|\_|$ (cf. Examples 3.4.8 and 3.4.39).

The "free group" functor $\mathbf{F}:\mathbf{Set} \to \mathbf{Grp}$ is left adjoint to the forgetful functor $|\_|:\mathbf{Grp} \to \mathbf{Set}$. Also, the functor taking a set X to the discrete topology on X is left adjoint to the forgetful functor $|\_|:\mathbf{Top} \to \mathbf{Set}$ (cf. Exercise 3.5.3). □

Exercise 3.5.11. Consider any algebraic signature morphism $\sigma:\Sigma \to \Sigma'$. Prove that the reduct functor $\_|_\sigma:\mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$ has a left adjoint.

HINT: Formalise and complete the following construction. For any Σ-algebra A, let $\Sigma(A)$ be an algebraic signature which extends Σ by a constant $a:s$ for each element $a \in |A|_s$, $s \in \mathit{sorts}(\Sigma)$, and let $\Sigma'(A)$ be a similar extension of Σ' by a constant $a:\sigma(s)$ for each $a \in |A|_s$, $s \in \mathit{sorts}(\Sigma)$. Consider the congruence $\equiv_A$ on $T_{\Sigma(A)}$ generated by the identities that hold in A. The congruence $\equiv_A$ may be translated by σ to $\Sigma'(A)$-terms, generating there a congruence $\sigma(\equiv_A)$, and the algebra $T_{\Sigma'(A)}/\sigma(\equiv_A)$ is (almost) the free Σ'-algebra over A.

Consider then a set Φ' of Σ'-equations. Recall that $\mathbf{Mod}(\Sigma', \Phi')$ is the full subcategory of $\mathbf{Alg}(\Sigma')$ with all Σ'-algebras that satisfy Φ' as objects (cf. Example 3.1.19). Prove that the reduct functor $\_|_\sigma:\mathbf{Mod}(\Sigma', \Phi') \to \mathbf{Alg}(\Sigma)$ has a left adjoint.

HINT: In the construction above, close the congruence $\sigma(\equiv_A)$ so that for each equation $\forall X' \bullet t = t'$ in Φ' and substitution $\theta:X' \to |T_{\Sigma'(A)}|$, it identifies the terms $t[\theta]$ and $t'[\theta]$ (cf. Exercise 1.4.9 for the notation used here).

Finally, for any set Φ of Σ-equations such that $\Phi' \models_{\Sigma'} \sigma(\Phi)$, prove that the reduct functor $\_|_\sigma : \mathbf{Mod}(\Sigma', \Phi') \to \mathbf{Mod}(\Sigma, \Phi)$ has a left adjoint.

HINT: This is easy now (Proposition 2.3.13 ensures that the functor is well defined). □

Exercise 3.5.12. Generalise Exercise 3.5.11 to derived signature morphisms, with reduct functors as introduced in Exercise 3.4.30. □

Example 3.5.13. Let $\mathbf{K}$ be a category, and recall that $\mathbf{1}$ is a category containing a single object, say a. Let $\mathbf{F} : \mathbf{1} \to \mathbf{K}$ be left adjoint to $\mathbf{C}_a : \mathbf{K} \to \mathbf{1}$ (note that such a functor $\mathbf{F}$ may not exist). Then $\mathbf{F}(a)$ is an initial object in $\mathbf{K}$. □

Exercise 3.5.14. Let $\Delta : \mathbf{K} \to \mathbf{K} \times \mathbf{K}$ be the "diagonal" functor such that $\Delta(A) = \langle A, A \rangle$ and $\Delta(f : A \to B) = \langle f, f \rangle : \Delta(A) \to \Delta(B)$. Prove that $\mathbf{K}$ has all coproducts iff Δ has a left adjoint. What is the unit? □

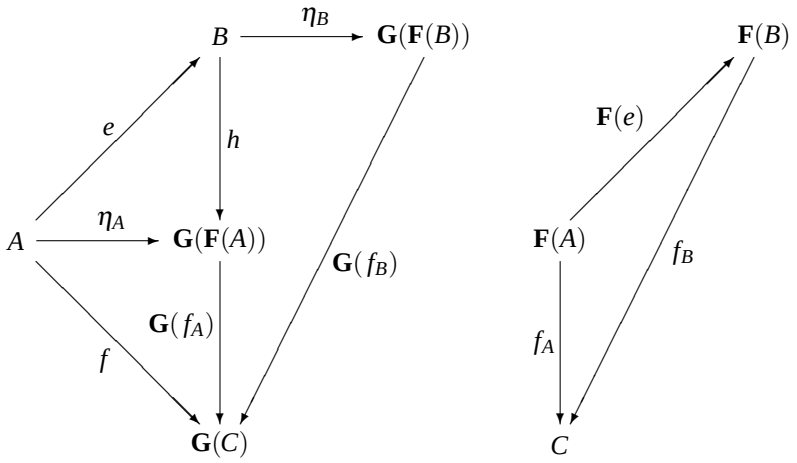
Exercise 3.5.15. Formulate analogous theorems for coequalisers and pushouts and prove them. Show how this may be done for any colimit. □

Exercise 3.5.16. Let $\mathbf{K}$ be a category with an initial object and a factorisation system and let $\mathbf{K}_R$ be its full subcategory of reachable objects. Recall that $\mathbf{R}_\mathbf{K} : \mathbf{K} \to \mathbf{K}_R$ is a functor that maps any object to its reachable subobject (cf. Exercise 3.4.13). Show that the inclusion functor $\mathbf{I} : \mathbf{K}_R \to \mathbf{K}$ is left adjoint to $\mathbf{R}_K$. □

Exercise 3.5.17. Show that left adjoints preserve colimits of diagrams. Do they preserve limits as well? □

Exercise 3.5.18. Let $\mathbf{F} : \mathbf{K2} \to \mathbf{K1}$ be left adjoint to $\mathbf{G} : \mathbf{K1} \to \mathbf{K2}$ with unit $\eta : \mathbf{Id}_\mathbf{K1} \to \mathbf{F};\mathbf{G}$. Consider two objects $A, B \in |\mathbf{K1}|$ and suppose that for some epimorphism $e : A \to B$ there exists a morphism $h : B \to \mathbf{G}(\mathbf{F}(A))$ such that $e;h = \eta_A$. Prove that $\mathbf{F}(e) : \mathbf{F}(A) \to \mathbf{F}(B)$ is an isomorphism.

HINT:



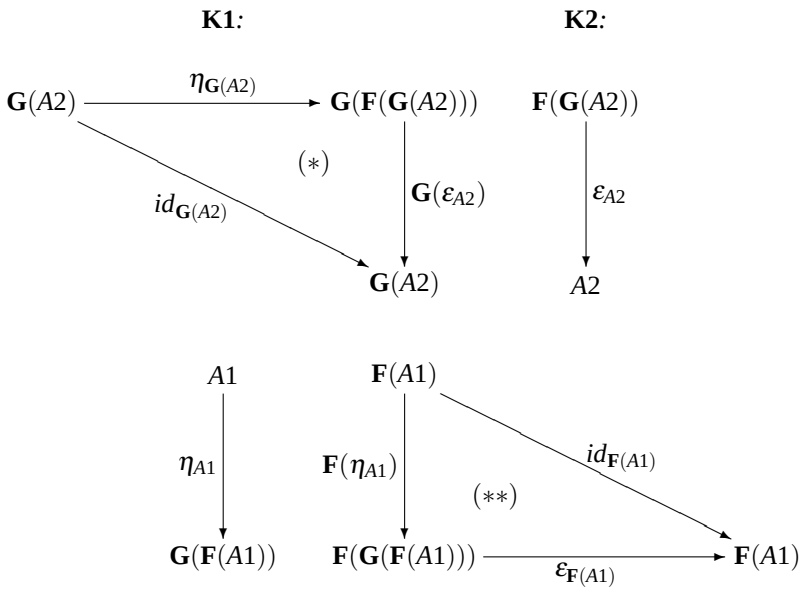
First show that $\mathbf{F}(B)$ with $e;\eta_B : A \to \mathbf{G}(\mathbf{F}(B))$ as the unit morphism is a free object over A w.r.t. $\mathbf{G}$. For this, use the following construction: for any $C \in |\mathbf{K2}|$ and $f : A \to \mathbf{G}(C)$, let $f_B : \mathbf{F}(B) \to C$ be the unique morphism such that $\eta_B;\mathbf{G}(f_B) = h;\mathbf{G}(f_A)$, where in turn $f_A : \mathbf{F}(A) \to C$ is the unique morphism such that $\eta_A;\mathbf{G}(f_A) = f$. Now, f_B satisfies $(e;\eta_B);\mathbf{G}(f_B) = f$ and moreover, it is the only morphism from $\mathbf{F}(B)$ to C with this property (use the fact that e is an epimorphism and the freeness of $\mathbf{F}(B)$ to prove the latter). Then, show the conclusion following the proof of the uniqueness of left adjoints, cf. Proposition 3.5.9. □

3.5.3 Adjunctions

Consider two categories **K1** and **K2** and functors $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G} : \mathbf{K2} \to \mathbf{K1}$ such that **F** is left adjoint to **G** with unit $\eta : \mathbf{Id_{K1}} \to \mathbf{F};\mathbf{G}$.

Proposition 3.5.19. *There is a natural transformation $\varepsilon : \mathbf{G};\mathbf{F} \to \mathbf{Id_{K2}}$ such that*

$$(*): \quad (\mathbf{G}\cdot\eta);(\varepsilon\cdot\mathbf{G}) = id_{\mathbf{G}}$$
$$(**): \quad (\eta\cdot\mathbf{F});(\mathbf{F}\cdot\varepsilon) = id_{\mathbf{F}}$$



Proof idea.

- $(*)$ defines $\varepsilon_{A2} : \mathbf{F}(\mathbf{G}(A2)) \to A2$ as $\varepsilon_{A2} = (id_{\mathbf{G}(A2)})^{\#}$.
- *Check naturality*: To show that for all $g : A2 \to B2$ in **K2**, $\varepsilon_{A2};g = \mathbf{F}(\mathbf{G}(g));\varepsilon_{B2}$, it is enough to prove that (in **K1**) $\eta_{\mathbf{G}(A2)};\mathbf{G}(\varepsilon_{A2};g) = \eta_{\mathbf{G}(A2)};\mathbf{G}(\mathbf{F}(\mathbf{G}(g));\varepsilon_{B2})$.
- *Check* $(**)$: To prove that $\mathbf{F}(\eta_{A1});\varepsilon_{\mathbf{F}(A1)} = id_{\mathbf{F}(A1)}$, it is enough to show that (in **K1**) $\eta_{A1};\mathbf{G}(\mathbf{F}(\eta_{A1});\varepsilon_{\mathbf{F}(A1)}) = \eta_{A1};\mathbf{G}(id_{\mathbf{F}(A1)})$. □

Proposition 3.5.20. *Consider functors* $\mathbf{F}:\mathbf{K1}\to\mathbf{K2}$ *and* $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$*, and natural transformations* $\eta:\mathbf{Id_{K1}}\to\mathbf{F};\mathbf{G}$ *and* $\varepsilon:\mathbf{G};\mathbf{F}\to\mathbf{Id_{K2}}$ *such that*

$$\begin{array}{rl}(*): & (\mathbf{G}\cdot\eta);(\varepsilon\cdot\mathbf{G})=id_{\mathbf{G}}\\ (**): & (\eta\cdot\mathbf{F});(\mathbf{F}\cdot\varepsilon)=id_{\mathbf{F}}\end{array}$$

Then $\mathbf{F}$ *is left adjoint to* $\mathbf{G}$ *with unit* η.

Proof. For $A1\in|\mathbf{K1}|$, $B2\in|\mathbf{K2}|$, $f:A1\to\mathbf{G}(B2)$, let $f^{\#}=\mathbf{F}(f);\varepsilon_{B2}:\mathbf{F}(A1)\to B2$.

- $\eta_{A1};\mathbf{G}(f^{\#})=\eta_{A1};\mathbf{G}(\mathbf{F}(f));\mathbf{G}(\varepsilon_{B2})=f;\eta_{\mathbf{G}(B2)};\mathbf{G}(\varepsilon_{B2})=f;id_{\mathbf{G}(B2)}=f$.
- Suppose that for some $g:\mathbf{F}(A1)\to B2$, $\eta_{A1};\mathbf{G}(g)=f$. Then: $f^{\#}=\mathbf{F}(f);\varepsilon_{B2}=\mathbf{F}(\eta_{A1};\mathbf{G}(g));\varepsilon_{B2}=\mathbf{F}(\eta_{A1});\mathbf{F}(\mathbf{G}(g));\varepsilon_{B2}=\mathbf{F}(\eta_{A1});\varepsilon_{\mathbf{F}(A1)};g=g$. □

Definition 3.5.21 (Adjunction). Let **K1** and **K2** be categories. An *adjunction from* **K1** *to* **K2** is a quadruple $\langle\mathbf{F},\mathbf{G},\eta,\varepsilon\rangle$ where $\mathbf{F}:\mathbf{K1}\to\mathbf{K2}$ and $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$ are functors and $\eta:\mathbf{Id_{K1}}\to\mathbf{F};\mathbf{G}$ and $\varepsilon:\mathbf{G};\mathbf{F}\to\mathbf{Id_{K2}}$ are natural transformations such that

$$\begin{array}{rl}(*): & (\mathbf{G}\cdot\eta);(\varepsilon\cdot\mathbf{G})=id_{\mathbf{G}}\\ (**): & (\eta\cdot\mathbf{F});(\mathbf{F}\cdot\varepsilon)=id_{\mathbf{F}}\end{array}$$ □

Fact 3.5.22. *Equivalently, an adjunction may be given as either of the following:*

- *A functor* $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$ *and for each* $A1\in|\mathbf{K1}|$*, a free object over* $A1$ *w.r.t.* $\mathbf{G}$*;*
- *A functor* $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$ *and its left adjoint.* □

Exercise 3.5.23 (Galois connection). Recall that any partial order gives rise to a corresponding preorder category (cf. Example 3.1.3). Galois connections (Definition 2.3.3) arise as adjunctions between preorder categories:

Consider two partially ordered sets $\langle A,\leq_A\rangle$ and $\langle B,\leq_B\rangle$ and two order-preserving functions $f:A\to B$ and $g:B\to A$ (i.e., for $a,a'\in A$, if $a\leq_A a'$ then $f(a)\leq_B f(a')$ and for $b,b'\in B$, if $b\leq_B b'$ then $g(b)\leq_A g(b')$).

Show that f and g (viewed as functors) form an adjunction between $\langle A,\leq_A\rangle$ and $\langle B,\leq_B\rangle$ (viewed as categories) if and only if for all $a\in A$ and $b\in B$:

$$a\leq_A g(b)\quad\text{iff}\quad f(a)\leq_B b$$

Then show that this is further equivalent to the requirement that:

- $a\leq_A g(f(a))$ for all $a\in A$; and
- $f(g(b))\leq_B b$ for all $b\in B$.

View the Galois connection between sets of equations and classes of algebras on a given signature defined in Section 2.3 (cf. Proposition 2.3.2) as a special case of the above definition. That is, check that for any signature Σ, the function mapping any set of Σ-equations to the class of all Σ-algebras that satisfy this set of equations and the function mapping any class of Σ-algebras to the set of all Σ-equations that hold in this class form an adjunction between the powerset of the set of Σ-equations (ordered by inclusion) and the powerclass of the class of Σ-algebras (ordered by containment).

Then check that the above definition of Galois connection coincides with the more explicit Definition 2.3.3 of a Galois connection between $\langle A, \leq_A \rangle$ and $\langle B, \geq_B \rangle$ (note the opposite order for B). □

Exercise 3.5.24. Dualise the development in this section. Begin with the following definition, dual to Definition 3.5.1:

Definition. Let $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ be a functor and let $A2 \in |\mathbf{K2}|$. A *cofree object over* $A2$ *w.r.t.* $\mathbf{F}$ is a $\mathbf{K1}$-object $A1$ together with a $\mathbf{K2}$-morphism $\varepsilon_{A2}:\mathbf{F}(A1) \to A2$ such that for any $\mathbf{K1}$-object $B1$ and $\mathbf{K2}$-morphism $f:\mathbf{F}(B1) \to A2$ there is a unique $\mathbf{K1}$-morphism $f^{\#}:B1 \to A1$ such that $\mathbf{F}(f^{\#});\varepsilon_{A2} = f$.

Then dually to Section 3.5.2 show how cofree objects induce *right adjoints*. Finally, prove facts dual to Propositions 3.5.19 and 3.5.20, thus proving that right adjoints and cofree objects give another equivalent definition of adjunction. □

Exercise 3.5.25. Develop yet another equivalent definition (at least for small categories) of an adjunction, centering around the bijection $\#:\mathbf{K1}(A1, \mathbf{G}(A2)) \to \mathbf{K2}(\mathbf{F}(A1), A2)$ using a generalised version of Hom-functors (cf. Example 3.4.15).

Proof sketch.

- For any small category $\mathbf{K}$ and two functors $\mathbf{F1}:\mathbf{K1} \to \mathbf{K}$ and $\mathbf{F2}:\mathbf{K2} \to \mathbf{K}$, define a functor $\mathbf{Hom}_{\mathbf{F1},\mathbf{F2}}:\mathbf{K1}^{op} \times \mathbf{K2} \to \mathbf{Set}$ by $\mathbf{Hom}_{\mathbf{F1},\mathbf{F2}}(\langle A1, A2 \rangle) = \mathbf{K}(\mathbf{F1}(A1), \mathbf{F2}(A2))$ and $\mathbf{Hom}_{\mathbf{F1},\mathbf{F2}}(\langle f1, f2 \rangle)(h) = \mathbf{F1}(f1);h;\mathbf{F2}(f2)$.
- Show that if $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ is left adjoint to $\mathbf{G}:\mathbf{K2} \to \mathbf{K1}$ then $\#:\mathbf{Hom}_{\mathbf{Id_{K1}},\mathbf{G}} \to \mathbf{Hom}_{\mathbf{F},\mathbf{Id_{K2}}}$ is a natural isomorphism.
- Finally, prove that for any functors $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G}:\mathbf{K2} \to \mathbf{K1}$, a natural isomorphism $\#:\mathbf{Hom}_{\mathbf{Id_{K1}},\mathbf{G}} \to \mathbf{Hom}_{\mathbf{F},\mathbf{Id_{K2}}}$ shows that $\mathbf{F}$ is left adjoint to $\mathbf{G}$. □

Exercise 3.5.26. Show that adjunctions compose: given any categories $\mathbf{K1}$, $\mathbf{K2}$ and $\mathbf{K3}$, and adjunctions $\langle \mathbf{F}, \mathbf{G}, \eta, \varepsilon \rangle$ from $\mathbf{K1}$ to $\mathbf{K2}$ and $\langle \mathbf{F}', \mathbf{G}', \eta', \varepsilon' \rangle$ from $\mathbf{K2}$ to $\mathbf{K3}$, we have an adjunction of the form $\langle \mathbf{F};\mathbf{F}', \mathbf{G}';\mathbf{G}, \_, \_ \rangle$ from $\mathbf{K1}$ to $\mathbf{K3}$. Fill in the holes! □

3.6 Bibliographical remarks

Category theory has found very many applications in computer science, and the material presented here covers just those fragments that we will require in later chapters. Books on category theory for mathematicians include the classic [Mac71] as well as the encyclopedic [HS73], with [AHS90] as a more recent favourite, the three-volume handbook [Bor94], the modestly-sized textbook [Awo06], and many more. An early book on category theory directed towards computer scientists is [AM75], followed by [Pie91], [Poi92] and [BW95]. An interesting angle is in [RB88], where categorical concepts are presented by coding them in ML.

Our terminology is mainly based on [Mac71], although we prefer to write composition in diagrammatic order, denoted by semicolon. The reader should be warned that the terminology and notation in category theory is not completely standardized, and differ from one author to another.

We have decided to keep to the basics, and have not ventured into many more advanced topics, some of which are quite important for computer-science applications. In particular, Cartesian closed categories [BW95], [Mit96] and the Curry-Howard isomorphism [SU06], categorical logic [LS86], monads [Man76], [Mog91], [Pho92], fibrations [Jac99], and topoi [Joh02], [Gol06] all deserve attention.

We have presented somewhat more material than usual on certain topics that will find application in some of the subsequent chapters. For example, in the material on factorisation systems (with Section 3.3 taken from [Tar85]) and on indexed categories (with Section 3.4.3.2 based on [TBG91]), we include some exercises which formulate facts that we will rely on later. We will work with indexed categories throughout the book, sometimes implicitly, since we find them more natural for these applications than equivalent formulations in terms of fibrations [Jac99].

We have deliberately chosen to use a notion of factorisation system based on [HS73]. The later book [AHS90] uses a somewhat more general concept, where factorisation morphisms are not required to be epi and mono, respectively, and therefore the uniqueness of the isomorphism between different factorisations of the same morphisms — or equivalently, of the diagonal in Lemma 3.3.4 — must be required explicitly. Although much of the material carries over, some results are simpler under our assumptions: for instance, we rely on Exercise 3.3.5 which does not hold in this form in the framework of [AHS90].

Our presentation of signatures, terms and algebras in Chapter 1 was elementary and set-theoretic, and we retain this style throughout the book. But category theory offers a whole spectrum of possibilities of doing universal algebra fruitfully in a different style. Exercises 3.4.26 and 3.4.41 relate to a categorical "Lawvere-style" presentation of some of the same concepts, see [Law63], [Man76], [BW85]. This was used in some early papers on algebraic specification, e.g. [GTWW75], but as it abstracts away from the choice of operation names in the signature, it seems less useful for applications to program specification. (This argument was put forward already in [BG80], with the notion of "signed theory" from [GB78] called to the rescue.) An alternative approach to specifications in this framework is given by sketches, see [BW95], which present specifications as graphs with indicated diagrams, cones and cocones that in a functorial model of the graph are mapped to commutative diagrams, limits and colimits, respectively. Commutative diagrams capture equational requirements here, with (co)limiting (co)cones offering additional specification power. Another related approach takes the general notion of a T-algebra for a functor $T : \mathbf{K} \to \mathbf{K}$ as its starting point, where a T-algebra on an object $A \in |\mathbf{K}|$ is a morphism from $T(A)$ to A; this works smoothly if T is a monad, see [Man76]. Such abstract approaches offer natural generalisations based on semantic interpretation in categories other than **Set**, but again, in our view, abstraction from familiar concepts and syntactic presentations makes them less convenient for practical use.

References

AC89. Egidio Astesiano and Maura Cerioli. On the existence of initial models for partial (higher-order) conditional specifications. In Josep Díaz and Fernando Orejas, editors, *Proceedings of the International Joint Conference on Theory and Practice of Software Development, TAPSOFT'89*, Barcelona, *Lecture Notes in Computer Science*, volume 351, pages 74–88. Springer, 1989.

AC01. David Aspinall and Adriana B. Compagnoni. Subtyping dependent types. *Theoretical Computer Science*, 266(1–2):273–309, 2001.

ACEGG91. Jaume Agustí-Cullell, Francesc Esteva, Pere Garcia, and Lluis Godo. Formalizing multiple-valued logics as institutions. In Bernadette Bouchon-Meunier, Ronald R. Yager, and Lotfi A. Zadeh, editors, *Proceedings of the 3rd International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU'90*, Paris, *Lecture Notes in Computer Science*, volume 521, pages 269–278. Springer, 1991.

AF96. Mário Arrais and José Luiz Fiadeiro. Unifying theories in different institutions. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification. Selected Papers from the 11th Workshop on Specification of Abstract Data Types*, Oslo, *Lecture Notes in Computer Science*, volume 1130, pages 81–101. Springer, 1996.

AG97. Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, 1997.

AH05. David Aspinall and Martin Hofmann. Dependent types. In Benjamin Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 2, pages 45–86. MIT Press, 2005.

AHS90. Jiři Adámek, Horst Herrlich, and George Strecker. *Abstract and Concrete Categories: The Joy of Cats*. Wiley, 1990.

Ala02. Suad Alagic. Institutions: Integrating objects, XML and databases. *Information and Software Technology*, 44(4):207–216, 2002.

AM75. Michael A. Arbib and Ernest G. Manes. *Arrows, Structures and Functors: The Categorical Imperative*. Academic Press, 1975.

Asp95. David Aspinall. Subtyping with singleton types. In Leszek Pacholski and Jerzy Tiuryn, editors, *Proceedings of the 8th International Workshop on Computer Science Logic, CSL'94*, Kazimierz, *Lecture Notes in Computer Science*, volume 933, pages 1–15. Springer, 1995.

Asp97. David Aspinall. *Type Systems for Modular Programming and Specification*. PhD thesis, University of Edinburgh, Department of Computer Science, 1997.

Asp00. David Aspinall. Subtyping with power types. In Peter Clote and Helmut Schwichtenberg, editors, *Proceedings of the 14th International Workshop on Computer Science*

Logic, Fischbachau, *Lecture Notes in Computer Science*, volume 1862, pages 156–171. Springer, 2000.

Avr91. Arnon Avron. Simple consequence relations. *Information and Computation*, 92:105–139, 1991.

Awo06. Steve Awodey. *Category Theory*. Oxford University Press, 2006.

Bar74. Jon Barwise. Axioms for abstract model theory. *Annals of Mathematical Logic*, 7:221–265, 1974.

BBB+85. Friedrich L. Bauer, Rudolf Berghammer, Manfred Broy, Walter Dosch, Franz Geiselbrechtinger, Rupert Gnatz, E. Hangel, Wolfgang Hesse, Bernd Krieg-Brückner, Alfred Laut, Thomas Matzner, Bernd Möller, Friederike Nickl, Helmut Partsch, Peter Pepper, Klaus Samelson, Martin Wirsing, and Hans Wössner. *The Munich Project CIP: Volume 1: The Wide Spectrum Language CIP-L*, *Lecture Notes in Computer Science*, volume 183. Springer, 1985.

BBC86. Gilles Bernot, Michel Bidoit, and Christine Choppy. Abstract data types with exception handling: An initial approach based on a distinction between exceptions and errors. *Theoretical Computer Science*, 46(1):13–45, 1986.

BC88. Val Breazu-Tannen and Thierry Coquand. Extensional models for polymorphism. *Theoretical Computer Science*, 59(1–2):85–114, 1988.

BCH99. Michel Bidoit, María Victoria Cengarle, and Rolf Hennicker. Proof systems for structured specifications and their refinements. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 11, pages 385–433. Springer, 1999.

BD77. R.M. Burstall and J. Darlington. A transformational system for developing recursive programs. *Journal of the Association for Computing Machinery*, 24(1):44–67, 1977.

BDP+79. Manfred Broy, Walter Dosch, Helmut Partsch, Peter Pepper, and Martin Wirsing. Existential quantifiers in abstract data types. In Hermann A. Maurer, editor, *Proceedings of the 6th International Colloquium on Automata, Languages and Programming*, Graz, *Lecture Notes in Computer Science*, volume 71, pages 73–87. Springer, 1979.

Bén85. Jean Bénabou. Fibred categories and the foundations of naïve category theory. *Journal of Symbolic Logic*, 50:10–37, 1985.

Ber87. Gilles Bernot. Good functors ... are those preserving philosophy! In David H. Pitt, Axel Poigné, and David E. Rydeheard, editors, *Proceedings of the 2nd Summer Conference on Category Theory and Computer Science*, Edinburgh, *Lecture Notes in Computer Science*, volume 283, pages 182–195. Springer, 1987.

BF85. Jon Barwise and Solomon Feferman, editors. *Model-Theoretic Logics*. Springer, 1985.

BG77. R.M. Burstall and J.A. Goguen. Putting theories together to make specifications. In *Fifth International Joint Conference on Artificial Intelligence*, pages 1045–1058, Boston, 1977.

BG80. R.M. Burstall and J.A. Goguen. The semantics of Clear, a specification language. In Dines Bjørner, editor, *Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification*, *Lecture Notes in Computer Science*, volume 86, pages 292–332. Springer, 1980.

BG81. R.M. Burstall and J.A. Goguen. An informal introduction to specifications using Clear. In R.S. Boyer and J.S. Moore, editors, *The Correctness Problem in Computer Science*, pages 185–213. Academic Press, 1981. Also in: *Software Specification Techniques* (eds. N. Gehani and A.D. McGettrick), Addison-Wesley, 1986.

BG01. Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

BH96. Michel Bidoit and Rolf Hennicker. Behavioural theories and the proof of behavioural properties. *Theoretical Computer Science*, 165(1):3–55, 1996.

BH98. Michel Bidoit and Rolf Hennicker. Modular correctness proofs of behavioural implementations. *Acta Informatica*, 35(11):951–1005, 1998.

BH06a. Michel Bidoit and Rolf Hennicker. Constructor-based observational logic. *Journal of Logic and Algebraic Programming*, 67(1–2):3–51, 2006.

BH06b. Michel Bidoit and Rolf Hennicker. Proving behavioral refinements of COL-specifications. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning and Computation: Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 4060, pages 333–354. Springer, 2006.

BHK90. Jan Bergstra, Jan Heering, and Paul Klint. Module algebra. *Journal of the Association for Computing Machinery*, 37(2):335–372, 1990.

BHW94. Michel Bidoit, Rolf Hennicker, and Martin Wirsing. Characterizing behavioural semantics and abstractor semantics. In Donald Sannella, editor, *Proceedings of the 5th European Symposium on Programming*, Edinburgh, *Lecture Notes in Computer Science*, volume 788, pages 105–119. Springer, 1994.

BHW95. Michel Bidoit, Rolf Hennicker, and Martin Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25(2-3):149–186, 1995.

Bir35. Garrett Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.

BL69. R.M. Burstall and P.J. Landin. Programs and their proofs: an algebraic approach. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 17–43. Edinburgh University Press, 1969.

BM04. Michel Bidoit and Peter D. Mosses, editors. CASL *User Manual*. Number 2900 in Lecture Notes in Computer Science. Springer, 2004.

BN98. Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

Bor94. Francis Borceaux. *Handbook of Categorical Algebra*. Cambridge University Press, 1994.

Bor00. Tomasz Borzyszkowski. Higher-order logic and theorem proving for structured specifications. In Didier Bert, Christine Choppy, and Peter D. Mosses, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 14th International Workshop on Algebraic Development Techniques*, Château de Bonas, *Lecture Notes in Computer Science*, volume 1827, pages 401–418. Springer, 2000.

Bor02. Tomasz Borzyszkowski. Logical systems for structured specifications. *Theoretical Computer Science*, 286(2):197–245, 2002.

Bor05. Tomasz Borzyszkowski. Generalized interpolation in first order logic. *Fundamenta Informaticae*, 66(3):199–219, 2005.

BPP85. Edward K. Blum and Francesco Parisi-Presicce. The semantics of shared submodules specifications. In Hartmut Ehrig, Christiane Floyd, Maurice Nivat, and James W. Thatcher, editors, *Mathematical Foundations of Software Development. Proceedings of the International Joint Conference on Theory and Practice of Software Development. Volume 1: Colloquium on Trees in Algebra and Programming*, *Lecture Notes in Computer Science*, volume 185, pages 359–373. Springer, 1985.

BRJ98. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

BS93. Rudolf Berghammer and Gunther Schmidt. Relational specifications. In C. Rauszer, editor, *Proc. XXXVIII Banach Center Semester on Algebraic Methods in Logic and their Computer Science Applications*, *Banach Center Publications*, volume 28, pages 167–190, Warszawa, 1993. Institute of Mathematics, Polish Academy of Sciences.

BST02. Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Architectural specifications in CASL. *Formal Aspects of Computing*, 13:252–273, 2002.

BST08. Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Observational interpretation of CASL specifications. *Mathematical Structures in Computer Science*, 18:325–371, 2008.

BT87. Jan Bergstra and John Tucker. Algebraic specifications of computable and semicomputable data types. *Theoretical Computer Science*, 50(2):137–181, 1987.

BT96. Michel Bidoit and Andrzej Tarlecki. Behavioural satisfaction and equivalence in concrete model categories. In Hélène Kirchner, editor, *Proceedings of the 21st International Colloquium on Trees in Algebra and Programming*, Linköping, *Lecture Notes in Computer Science*, volume 1059, pages 241–256. Springer, 1996.

Bur86. Peter Burmeister. *A Model Theoretic Oriented Approach to Partial Algebras*. Akademie-Verlag, 1986.

BW82a. Friedrich L. Bauer and Hans Wössner. *Algorithmic Language and Program Development*. Springer, 1982.

BW82b. Manfred Broy and Martin Wirsing. Partial abstract data types. *Acta Informatica*, 18(1):47–64, 1982.

BW85. Michael Barr and Charles Wells. *Toposes, Triples and Theories*. Number 278 in Grundlehren der mathematischen Wissenschaften. Springer, 1985.

BW95. Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice Hall, second edition, 1995.

BWP84. Manfred Broy, Martin Wirsing, and Claude Pair. A systematic study of models of abstract data types. *Theoretical Computer Science*, 33(2–3):139–174, 1984.

Car88. Luca Cardelli. Structural subtyping and the notion of power type. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages*, San Diego, pages 70–79, 1988.

CDE+02. Manuel Clavela, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, 2002. See also `http://maude.cs.uiuc.edu/`.

Cen94. María Victoria Cengarle. *Formal Specifications with Higher-Order Parameterization*. PhD thesis, Ludwig-Maximilians-Universität München, Institut für Informatik, 1994.

CF92. Robin Cockett and Tom Fukushima. About Charity. Technical Report No. 92/480/18, Department of Computer Science, University of Calgary, 1992.

CGR03. Carlos Caleiro, Paula Gouveia, and Jaime Ramos. Completeness results for fibred parchments: Beyond the propositional base. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 16th International Workshop on Algebraic Development Techniques*, Frauenchiemsee, *Lecture Notes in Computer Science*, volume 2755, pages 185–200. Springer, 2003.

Chu56. Alonzo Church. *Introduction to Mathematical Logic, Volume 1*. Princeton University Press, 1956.

Cîr02. Corina Cîrstea. On specification logics for algebra-coalgebra structures: Reconciling reachability and observability. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures. European Joint Conferences on Theory and Practice of Software (ETAPS 2002)*, Grenoble, *Lecture Notes in Computer Science*, volume 2303, pages 82–97. Springer, 2002.

CJ95. Aurelio Carboni and Peter Johnstone. Connected limits, familial representability and Artin glueing. *Mathematical Structures in Computer Science*, 5(4):441–459, 1995.

CK90. Chen-Chung Chang and H. Jerome Keisler. *Model Theory*. North-Holland, third edition, 1990.

CK08a. María Victoria Cengarle and Alexander Knapp. An institution for OCL 2.0. Technical Report I0801, Institut für Informatik, Ludwig-Maximilians-Universität München, 2008.

CK08b. María Victoria Cengarle and Alexander Knapp. An institution for UML 2.0 interactions. Technical Report I0808, Institut für Informatik, Ludwig-Maximilians-Universität München, 2008.

CK08c. María Victoria Cengarle and Alexander Knapp. An institution for UML 2.0 static structures. Technical Report I0807, Institut für Informatik, Ludwig-Maximilians-Universität München, 2008.

CKTW08. Maria-Victoria Cengarle, Alexander Knapp, Andrzej Tarlecki, and Martin Wirsing. A heterogeneous approach to UML semantics. In Pierpaolo Degano, Rocco de Nicola, and José Meseguer, editors, *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 5065, pages 383–402. Springer, 2008.

CM97. Maura Cerioli and José Meseguer. May I borrow your logic? (Transporting logical structures along maps). *Theoretical Computer Science*, 173(2):311–347, 1997.

CMRM10. Mihai Codescu, Till Mossakowski, Adrían Riesco, and Christian Maeder. Integrating Maude into Hets. In Mike Johnson and Dusko Pavlovic, editors, *AMAST 2010*, Lecture Notes in Computer Science. Springer, 2010.

CMRS01. Carlos Caleiro, Paulo Mateus, Jaime Ramos, and Amílcar Sernadas. Combining logics: Parchments revisited. In Maura Cerioli and Gianna Reggio, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 15th Workshop on Algebraic Development Techniques joint with the CoFI WG Meeting*, Genova, *Lecture Notes in Computer Science*, volume 2267, pages 48–70. Springer, 2001.

Coh65. Paul M. Cohn. *Universal Algebra*. Harper and Row, 1965.

CS92. Robin Cockett and Dwight Spencer. Strong categorical datatypes I. In R.A.G. Seely, editor, *International Meeting on Category Theory 1991*, Canadian Mathematical Society Proceedings. American Mathematical Society, 1992.

CSS05. Carlos Caleiro, Amílcar Sernadas, and Cristina Sernadas. Fibring logics: Past, present and future. In Sergei N. Artemov, Howard Barringer, Artur S. d'Avila Garcez, Luís C. Lamb, and John Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay, Volume One*, pages 363–388. College Publications, 2005.

DF98. Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, *AMAST Series in Computing*, volume 6. World Scientific, 1998.

DF02. Răzvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. *Theoretical Computer Science*, 285:289–318, 2002.

DGS93. Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gérard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge University Press, 1993.

Dia00. Răzvan Diaconescu. Category-based constraint logic. *Mathematical Structures in Computer Science*, 10(3):373–407, 2000.

Dia02. Răzvan Diaconescu. Grothendieck institutions. *Applied Categorical Structures*, 10(4):383–402, 2002.

Dia08. Răzvan Diaconescu. *Institution-independent Model Theory*. Birkhäuser, 2008.

DJ90. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B (Formal Models and Semantics)*, pages 244–320. North-Holland and MIT Press, 1990.

DLL62. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

DM00. Theodosis Dimitrakos and Tom Maibaum. On a generalised modularisation theorem. *Information Processing Letters*, 74(1–2):65–71, 2000.

DMR76. Martin Davis, Yuri Matiyasevich, and Julia Robinson. Hilbert's tenth problem. Diophantine equations: Positive aspects of a negative solution. In *Mathematical Developments Arising from Hilbert Problems*, *Proceedings of Symposia in Pure Mathematics*, volume 28, pages 323–378, Providence, Rhode Island, 1976. American Mathematical Society.

DP90. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

Ehr78. Hans-Dieter Ehrich. Extensions and implementations of abstract data type specifications. In Józef Winkowski, editor, *Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science*, Zakopane, *Lecture Notes in Computer Science*, volume 64, pages 155–164. Springer, 1978.

Ehr81. Hans-Dieter Ehrich. On realization and implementation. In Jozef Gruska and Michal Chytil, editors, *Proceedings of the 10th Symposium on Mathematical Foundations of Computer Science*, Štrbské Pleso, *Lecture Notes in Computer Science*, volume 118, pages 271–280. Springer, 1981.

Ehr82. Hans-Dieter Ehrich. On the theory of specification, implementation and parametrization of abstract data types. *Journal of the Association for Computing Machinery*, 29(1):206–227, 1982.

EKMP82. Hartmut Ehrig, Hans-Jörg Kreowski, Bernd Mahr, and Peter Padawitz. Algebraic implementation of abstract data types. *Theoretical Computer Science*, 20:209–263, 1982.

EKT+80. Hartmut Ehrig, Hans-Jörg Kreowski, James Thatcher, Eric Wagner, and Jesse Wright. Parameter passing in algebraic specification languages. Technical report, Technische Universität Berlin, 1980.

EKT+83. Hartmut Ehrig, Hans-Jörg Kreowski, James Thatcher, Eric Wagner, and Jesse Wright. Parameter passing in algebraic specification languages. *Theoretical Computer Science*, 28(1–2):45–81, 1983.

EM85. Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1*, *EATCS Monographs on Theoretical Computer Science*, volume 6. Springer, 1985.

Eme90. E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B (Formal Models and Semantics)*, pages 995–1072. North-Holland and MIT Press, 1990.

End72. Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

EPO89. Hartmut Ehrig, Peter Pepper, and Fernando Orejas. On recent trends in algebraic specification. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *Proceeding of the 16th International Colloquium on Automata, Languages and Programming*, Stresa, *Lecture Notes in Computer Science*, volume 372, pages 263–288. Springer, 1989.

EWT83. Hartmut Ehrig, Eric G. Wagner, and James W. Thatcher. Algebraic specifications with generating constraints. In *Proceeding of the 10th International Colloquium on Automata, Languages and Programming*, Barcelona, *Lecture Notes in Computer Science*, volume 154, pages 188–202. Springer, 1983.

Far89. Jordi Farrés-Casals. Proving correctness of constructor implementations. In Antoni Kreczmar and Grazyna Mirkowska, editors, *Proceedings of the 14th Symposium on Mathematical Foundations of Computer Science*, Porabka-Kozubnik, *Lecture Notes in Computer Science*, volume 379, pages 225–235. Springer, 1989.

Far90. Jordi Farrés-Casals. Proving correctness wrt specifications with hidden parts. In Hélène Kirchner and Wolfgang Wechler, editors, *Proceedings of the 2nd International Conference on Algebraic and Logic Programming*, Nancy, *Lecture Notes in Computer Science*, volume 463, pages 25–39. Springer, 1990.

Far92. Jordi Farrés-Casals. *Verification in ASL and Related Specification Languages*. PhD thesis, University of Edinburgh, Department of Computer Science, 1992.

FC96. José Luiz Fiadeiro and José Félix Costa. Mirror, mirror in my hand: A duality between specifications and models of process behaviour. *Mathematical Structures in Computer Science*, 6(4):353–373, 1996.

Fei89. Loe M. G. Feijs. The calculus $\lambda\pi$. In Martin Wirsing and Jan A. Bergstra, editors, *Proceedings of the Workshop on Algebraic Methods: Theory, Tools and Applications*, *Lecture Notes in Computer Science*, volume 394, pages 307–328. Springer, 1989.

FGT92. William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. Little theories. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, *Lecture Notes in Artificial Intelligence*, volume 607, pages 567–581, Saratoga Springs, 1992. Springer.

Fia05. José Luiz Fiadeiro. *Categories for Software Engineering*. Springer, 2005.

Fit08. John S. Fitzgerald. The typed logic of partial functions and the Vienna Development Method. In Dines Bjørner and Martin Henson, editors, *Logics of Specification Languages*, pages 453–487. Springer, 2008.

FJ90. J. Fitzgerald and C.B. Jones. Modularizing the formal description of a database system. In *Proceedings of the 3rd International Symposium of VDM Europe: VDM and Z, Formal Methods in Software Development*, Kiel, *Lecture Notes in Computer Science*, volume 428, pages 189–210. Springer, 1990.

FS88. José Luiz Fiadeiro and Amílcar Sernadas. Structuring theories on consequence. In Donald Sannella and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification. Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, *Lecture Notes in Computer Science*, volume 332, pages 44–72. Springer, 1988.

Gab98. Dov M. Gabbay. *Fibring Logics*, *Oxford Logic Guides*, volume 38. Oxford University Press, 1998.

Gan83. Harald Ganzinger. Parameterized specifications: Parameter passing and implementation with respect to observability. *ACM Transactions on Programming Languages and Systems*, 5(3):318–354, 1983.

GB78. J.A. Goguen and R.M. Burstall. Some fundamental properties of algebraic theories: a tool for semantics of computation. Technical Report 53, Department of Artificial Intelligence, University of Edinburgh, 1978. Revised version appeared as [GB84b] and [GB84c].

GB80. J.A. Goguen and R.M. Burstall. CAT, a system for the structured elaboration of correct programs from structured specifications. Technical Report CSL-118, Computer Science Laboratory, SRI International, 1980.

GB84a. J.A. Goguen and R.M. Burstall. Introducing institutions. In Edmund Clarke and Dexter Kozen, editors, *Proceedings of the Workshop on Logics of Programs*, Pittsburgh, *Lecture Notes in Computer Science*, volume 164, pages 221–256. Springer, 1984.

GB84b. J.A. Goguen and R.M. Burstall. Some fundamental algebraic tools for the semantics of computation. Part 1: Comma categories, colimits, signatures and theories. *Theoretical Computer Science*, 31:175–209, 1984.

GB84c. J.A. Goguen and R.M. Burstall. Some fundamental algebraic tools for the semantics of computation. Part 2: Signed and abstract theories. *Theoretical Computer Science*, 31:263–295, 1984.

GB86. Joseph A. Goguen and Rod M. Burstall. A study in the functions of programming methodology: Specifications, institutions, charters and parchments. In David H. Pitt, Samson Abramsky, Axel Poigné, and David E. Rydeheard, editors, *Proceedings of the Tutorial and Workshop on Category Theory and Computer Programming*, Guildford, *Lecture Notes in Computer Science*, volume 240, pages 313–333. Springer, 1986.

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

GD94a. Joseph Goguen and Răzvan Diaconescu. An Oxford survey of order sorted algebra. *Mathematical Structures in Computer Science*, 4(3):363–392, 1994.

GD94b. Joseph A. Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Hartmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification. Selected Papers from the 9th Workshop on Specification of Abstract Data Types joint with the 4th* COMPASS *Workshop*, Caldes de Malavella, *Lecture Notes in Computer Science*, volume 785, pages 1–29. Springer, 1994.

GDLE84. Martin Gogolla, Klaus Drosten, Udo Lipeck, and Hans-Dieter Ehrich. Algebraic and operational semantics of specifications allowing exceptions and errors. *Theoretical Computer Science*, 34(3):289–313, 1984.

GG89. Stephen J. Garland and John V. Guttag. An overview of LP, the Larch Prover. In *Third International Conference on Rewriting Techniques and Applications*, Chapel Hill, *Lecture Notes in Computer Science*, volume 355, pages 137–151. Springer, 1989. See also `http://nms.lcs.mit.edu/larch/LP/all.html`.

GGM76. V. Giarratana, F. Gimona, and Ugo Montanari. Observability concepts in abstract data type specifications. In Antoni Mazurkiewicz, editor, *Proceedings of the 5th Sympo-*

sium on Mathematical Foundations of Computer Science, Gdańsk, *Lecture Notes in Computer Science*, volume 45, pages 567–578. Springer, 1976.

GH78. John Guttag and James Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

GH93. John V. Guttag and James J. Horning. *Larch: Languages and Tools for Formal Specification*. Springer, 1993.

Gin68. Abraham Ginzburg. *Algebraic Theory of Automata*. Academic Press, 1968.

Gir87. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

Gir89. Jean-Yves Girard. *Proofs and Types*, *Cambridge Tracts in Theoretical Computer Science*, volume 7. Cambridge University Press, 1989. Translated and with appendices by Paul Taylor and Yves Lafont.

GLR00. Joseph Goguen, Kai Lin, and Grigore Roşu. Circular coinductive rewriting. In *Proceedings of the 15th International Conference on Automated Software Engineering*, Grenoble. IEEE Computer Society, 2000.

GM82. Joseph A. Goguen and José Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In Mogens Nielsen and Erik Meineche Schmidt, editors, *Proceeding of the 9th International Colloquium on Automata, Languages and Programming*, Aarhus, *Lecture Notes in Computer Science*, volume 140, pages 265–281. Springer, 1982.

GM85. Joseph Goguen and José Meseguer. Completeness of many sorted equational deduction. *Houston Journal of Mathematics*, 11(3):307–334, 1985.

GM92. Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.

GM00. Joseph A. Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000.

Gog73. Joseph Goguen. Categorical foundations for general systems theory. In F. Pichler and R. Trappl, editors, *Advances in Cybernetics and Systems Research*, London, pages 121–130. Transcripta Books, 1973.

Gog74. J.A. Goguen. Semantics of computation. In Ernest G. Manes, editor, *Proceedings of the 1st International Symposium on Category Theory Applied to Computation and Control*, San Francisco, *Lecture Notes in Computer Science*, volume 25, pages 151–163. Springer, 1974.

Gog78. Joseph Goguen. Abstract errors for abstract data types. In Erich Neuhold, editor, *Formal Description of Programming Concepts*, pages 491–526. North-Holland, 1978.

Gog84. Martin Gogolla. Partially ordered sorts in algebraic specifications. In *Proceedings of the 9th Colloquium on Trees in Algebra and Programming*, pages 139–153. Cambridge University Press, 1984.

Gog85. Martin Gogolla. A final algebra semantics for errors and exceptions. In Hans-Jörg Kreowski, editor, *Recent Trends in Data Type Specification. Selected Papers from the 3rd Workshop on Theory and Applications of Abstract Data Types*, Bremen, *Informatik-Fachberichte*, volume 116, pages 89–103. Springer, 1985.

Gog91a. Joseph Goguen. Types as theories. In G.M. Reed, A.W. Roscoe, and R.F. Wachter, editors, *Topology and Category Theory in Computer Science*, Oxford, pages 357–390. Oxford University Press, 1991.

Gog91b. Joseph A. Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67, 1991.

Gog96. Joseph A. Goguen. Parameterized programming and software architecture. In Murali Sitaraman, editor, *Proceedings of the Fourth International Conference on Software Reuse*, pages 2–11. IEEE Computer Society Press, 1996.

Gog10. Joseph Goguen. Information integration in institutions. In Larry Moss, editor, *Thinking Logically: a Volume in Memory of Jon Barwise*. CSLI, Stanford University, 2010. To appear.

Gol06. Robert Goldblatt. *Topoi: The Categorial Analysis of Logic*. Dover, revised edition, 2006.

Gor95. Andrew D. Gordon. Bisimilarity as a theory of functional programming. In *Proceedings of the 11th Annual Conference on Mathematical Foundations of Programming Semantics*. *Electronic Notes in Theoretical Computer Science*, 1:232–252, 1995.

GR02. Joseph A. Goguen and Grigore Roşu. Institution morphisms. *Formal Aspects of Computing*, 13(3-5):274–307, 2002.

GR04. Joseph A. Goguen and Grigore Roşu. Composing hidden information modules over inclusive institutions. In *From Object-Orientation to Formal Methods. Essays in Memory of Ole-Johan Dahl*, *Lecture Notes in Computer Science*, volume 2635, pages 96–123. Springer, 2004.

Grä79. George A. Grätzer. *Universal Algebra*. Springer, second edition, 1979.

GS90. Carl Gunter and Dana Scott. Semantic domains. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B (Formal Models and Semantics)*, pages 633–674. North-Holland and MIT Press, 1990.

GTW76. Joseph Goguen, James Thatcher, and Eric Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. Technical Report RC 6487, IBM Watson Research Center, Yorktown Heights NY, 1976. Also in: *Current Trends in Programming Methodology. Volume IV (Data Structuring)* (ed. R.T. Yeh), Prentice-Hall, 80–149, 1978.

GTWW73. Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. A junction between computer science and category theory, I: Basic concepts and examples (part 1). Technical Report RC 4526, IBM Watson Research Center, Yorktown Heights NY, 1973.

GTWW75. Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. An introduction to categories, algebraic theories and algebras. Technical Report RC 5369, IBM Watson Research Center, Yorktown Heights NY, 1975.

GTWW77. Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, 24(1):68–95, 1977.

Gut75. John Guttag. *The Specification and Application to Programming of Abstract Data Types*. PhD thesis, University of Toronto, Department of Computer Science, 1975.

Hag87. Tatsuya Hagino. *A Categorical Programming Language*. PhD thesis, University of Edinburgh, Department of Computer Science, 1987.

Häh01. Reiner Hähnle. Tableaux and related methods. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 100–178. Elsevier and MIT Press, 2001.

Hal70. Paul R. Halmos. *Naive Set Theory*. Undergraduate Texts in Mathematics. Springer, 1970.

Hat82. William Hatcher. *The Logical Foundations of Mathematics*. Foundations and Philosophy of Science and Technology. Pergamon Press, 1982.

Hay94. Susumu Hayashi. Singleton, union and intersection types for program extraction. *Information and Computation*, 109(1/2):174–210, 1994.

Hee86. Jan Heering. Partial evaluation and ω-completeness of algebraic specifications. *Theoretical Computer Science*, 43:149–167, 1986.

Hen91. Rolf Hennicker. Context induction: A proof principle for behavioural abstractions and algebraic implementations. *Formal Aspects of Computing*, 3(4):326–345, 1991.

HHP93. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

HHWT97. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

Hig63. Phillip J. Higgins. Algebras with a scheme of operators. *Mathematische Nachrichten*, 27:115–132, 1963.

HLST00. Furio Honsell, John Longley, Donald Sannella, and Andrzej Tarlecki. Constructive data refinement in typed lambda calculus. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures. European Joint Conferences on Theory and Practice of Software (ETAPS 2000)*, Berlin, *Lecture Notes in Computer Science*, volume 1784, pages 161–176. Springer, 2000.

Hoa72. C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

HS73. Horst Herrlich and George E. Strecker. *Category Theory: An Introduction*. Allyn and Bacon, 1973.

HS96. Martin Hofmann and Donald Sannella. On behavioural abstraction and behavioural satisfaction in higher-order logic. *Theoretical Computer Science*, 167:3–45, 1996.

HS02. Furio Honsell and Donald Sannella. Prelogical relations. *Information and Computation*, 178:23–43, 2002.

HST94. Robert Harper, Donald Sannella, and Andrzej Tarlecki. Structured presentations and logic representations. *Annals of Pure and Applied Logic*, 67:113–160, 1994.

Hus92. Heinrich Hussmann. Nondeterministic algebraic specifications and nonconfluent term rewriting. *Journal of Logic Programming*, 12(1–4):237–255, 1992.

HWB97. Rolf Hennicker, Martin Wirsing, and Michel Bidoit. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173(2):393–443, 1997.

Jac99. Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1999.

JL87. Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, Munich, pages 111–119, 1987.

JNW96. André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

JOE95. Rosa M. Jiménez, Fernando Orejas, and Hartmut Ehrig. Compositionality and compatibility of parameterization and parameter passing in specification languages. *Mathematical Structures in Computer Science*, 5(2):283–314, 1995.

Joh02. Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Oxford Logic Guides Series. Clarendon Press, 2002.

Jon80. Cliff B. Jones. *Software Development: A Rigorous Approach*. Prentice-Hall, 1980.

Jon89. Hans B.M. Jonkers. An introduction to COLD-K. In Martin Wirsing and Jan A. Bergstra, editors, *Proceedings of the Workshop on Algebraic Methods: Theory, Tools and Applications*, *Lecture Notes in Computer Science*, volume 394, pages 139–205. Springer, 1989.

JR97. Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Computer Science*, 62:222–259, 1997.

KB70. Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

Kir99. Hélène Kirchner. Term rewriting. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 9, pages 273–320. Springer, 1999.

KKM88. Claude Kirchner, Hélène Kirchner, and José Meseguer. Operational semantics of OBJ-3. In Timo Lepistö and Arto Salomaa, editors, *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, Tampere, *Lecture Notes in Computer Science*, volume 317, pages 287–301. Springer, 1988.

Klo92. Jan Klop. Term rewriting systems. In Samson Abramsky, Dov Gabbay, and Tom Maibaum, editors, *Handbook of Logic in Computer Science. Volume 2 (Background: Computational Structures)*, pages 1–116. Oxford University Press, 1992.

KM87. Deepak Kapur and David R. Musser. Proof by consistency. *Artificial Intelligence*, 31(2):125–157, 1987.

KR71. Heinz Kaphengst and Horst Reichel. Algebraische Algorithmentheorie. Technical Report WIB 1, VEB Robotron, Zentrum für Forschung und Technik, Dresden, 1971.

Kre87. Hans-Jörg Kreowski. Partial algebras flow from algebraic specifications. In T. Ottmann, editor, *Proceedings of the 14th International Colloquium on Automata, Languages and Programming*, Karlsruhe, *Lecture Notes in Computer Science*, volume 267, pages 521–530. Springer, 1987.

KST97. Stefan Kahrs, Donald Sannella, and Andrzej Tarlecki. The definition of Extended ML: A gentle introduction. *Theoretical Computer Science*, 173:445–484, 1997.

KTB91. Beata Konikowska, Andrzej Tarlecki, and Andrzej Blikle. A three-valued logic for software specification and validation. *Fundamenta Informaticae*, 14(4):411–453, 1991.

Las98. Sławomir Lasota. Open maps as a bridge between algebraic observational equivalence and bisimilarity. In Francesco Parisi-Presicce, editor, *Recent Trends in Data Type Specification. Selected Papers from the 12th International Workshop on Specification of Abstract Data Types*, Tarquinia, *Lecture Notes in Computer Science*, volume 1376, pages 285–299. Springer, 1998.

Law63. F. William Lawvere. *Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963.

LB88. Butler Lampson and Rod Burstall. Pebble, a kernel language for modules and abstract data types. *Information and Computation*, 76(2/3):278–346, 1988.

LEW96. Jacques Loeckx, Hans-Dieter Ehrich, and Markus Wolf. *Specification of Abstract Data Types*. John Wiley and Sons, 1996.

Lin03. Kai Lin. *Machine Support for Behavioral Algebraic Specification and Verification*. PhD thesis, University of California, San Diego, 2003.

Lip83. Udo Lipeck. *Ein algebraischer Kalkül für einen strukturierten Entwurf von Datenabstraktionen*. PhD thesis, Universität Dortmund, 1983.

LLD06. Dorel Lucanu, Yuan-Fang Li, and Jin Song Dong. Semantic Web languages—towards an institutional perspective. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning and Computation: Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 4060, pages 99–123. Springer, 2006.

LS86. Joachim Lambek and Philip J. Scott. *Introduction to Higher-Order Categorical Logic*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1986.

LS00. Hugo Lourenço and Amílcar Sernadas. An institution of hybrid systems. In Didier Bert, Christine Choppy, and Peter D. Mosses, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 14th International Workshop on Algebraic Development Techniques*, Château de Bonas, *Lecture Notes in Computer Science*, volume 1827, pages 219–236. Springer, 2000.

Luo93. Zhaohui Luo. Program specification and data refinement in type theory. *Mathematical Structures in Computer Science*, 3(3):333–363, 1993.

Mac71. Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

Mac84. David B. MacQueen. Modules for Standard ML. In *Proceedings of the 1984 ACM Conference on LISP and Functional Programming*, pages 198–207, 1984.

MAH06. Till Mossakowski, Serge Autexier, and Dieter Hutter. Development graphs — proof management for structured specifications. *Journal of Logic and Algebraic Programming*, 67(1–2):114–145, 2006.

Mai72. Tom Maibaum. The characterization of the derivation trees of context free sets of terms as regular sets. In *Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory*, pages 224–230, 1972.

Maj77. Mila E. Majster. Limits of the "algebraic" specification of abstract data types. *ACM SIGPLAN Notices*, 12(10):37–42, 1977.

Mal71. Anatoly Malcev. Quasiprimitive classes of abstract algebras in the metamathematics of algebraic systems. In *Mathematics of Algebraic Systems: Collected Papers, 1936-67*, number 66 in Studies in Logic and Mathematics, pages 27–31. North-Holland, 1971.

Man76. Ernest G. Manes. *Algebraic Theories*. Springer, 1976.

May85. Brian Mayoh. Galleries and institutions. Technical Report DAIMI PB-191, Aarhus University, 1985.

Mei92. Karl Meinke. Universal algebra in higher types. *Theoretical Computer Science*, 100:385–417, 1992.

Mes89. José Meseguer. General logics. In H.-D. Ebbinghaus, editor, *Logic Colloquium '87*, Granada, pages 275–329. North-Holland, 1989.

Mes92. José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

Mes98. José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi-Presicce, editor, *Recent Trends in Data Type Specification. Selected Papers from the 12th International Workshop on Specification of Abstract Data Types*, Tarquinia, *Lecture Notes in Computer Science*, volume 1376, pages 18–61. Springer, 1998.

Mes09. José Meseguer. Order-sorted parameterization and induction. In Jens Palsberg, editor, *Semantics and Algebraic Specification: Essays Dedicated to Peter D. Mosses on the Occasion of His 60th Birthday*, *Lecture Notes in Computer Science*, volume 5700, pages 43–80. Springer, 2009.

MG85. José Meseguer and Joseph Goguen. Initiality, induction and computability. In Maurice Nivat and John C. Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge, 1985.

MGDT07. Till Mossakowski, Joseph Goguen, Răzvan Diaconescu, and Andrzej Tarlecki. What is a logic? In Jean-Yves Beziau, editor, *Logica Universalis: Towards a General Theory of Logic*, pages 111–135. Birkhäuser, 2007.

MHST08. Till Mossakowski, Anne Haxthausen, Donald Sannella, and Andrzej Tarlecki. CASL — the common algebraic specification language. In Dines Bjørner and Martin Henson, editors, *Logics of Specification Languages*, pages 241–298. Springer, 2008.

Mid93. Aart Middeldorp. Modular properties of conditional term rewriting systems. *Information and Computation*, 104(1):110–158, 1993.

Mil71. Robin Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489, 1971.

Mil77. Robin Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4(1):1–22, 1977.

Mil89. Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

Mit96. John C. Mitchell. *Foundations of Programming Languages*. MIT Press, 1996.

MM84. Bernd Mahr and Johann Makowsky. Characterizing specification languages which admit initial semantics. *Theoretical Computer Science*, 31:49–60, 1984.

MML07. Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, HETS. In Orna Grumberg and Michael Huth, editors, *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. European Joint Conferences on Theory and Practice of Software (ETAPS 2007)*, Braga, *Lecture Notes in Computer Science*, volume 4424, pages 519–522. Springer, 2007. See also `http://www.informatik.uni-bremen.de/cofi/hets/`.

Mog91. Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

Moo56. Edward F. Moore. Gedanken-experiments on sequential machines. In Claude E. Shannon and John McCarthy, editors, *Annals of Mathematics Studies 34, Automata Studies*, pages 129–153. Princeton University Press, 1956.

Mos89. Peter D. Mosses. Unified algebras and modules. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*, Austin, pages 329–343, 1989.

Mos93. Peter Mosses. The use of sorts in algebraic specifications. In Michel Bidoit and Christine Choppy, editors, *Recent Trends in Data Type Specification. Selected Papers from the 8th Workshop on Specification of Abstract Data Types joint with the 3rd* COMPASS *Workshop*, Dourdan, *Lecture Notes in Computer Science*, volume 655, pages 66–91. Springer, 1993.

Mos96a. Till Mossakowski. Different types of arrow between logical frameworks. In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *Proceedings of the 23rd International Colloquium Automata, Languages and Programming*, Paderborn, *Lecture Notes in Computer Science*, volume 1099, pages 158–169. Springer, 1996.

Mos96b. Till Mossakowski. *Representations, Hierarchies and Graphs of Institutions*. PhD thesis, Universität Bremen, 1996.

Mos00. Till Mossakowski. Specification in an arbitrary institution with symbols. In Didier Bert, Christine Choppy, and Peter D. Mosses, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 14th International Workshop on Algebraic Development Techniques*, Château de Bonas, *Lecture Notes in Computer Science*, volume 1827, pages 252–270. Springer, 2000.

Mos02. Till Mossakowski. Comorphism-based Grothendieck logics. In Krzysztof Diks and Wojciech Rytter, editors, *Proceedings of the 27th Symposium on Mathematical Foundations of Computer Science*, Warsaw, *Lecture Notes in Computer Science*, volume 2420, pages 593–604. Springer, 2002.

Mos03. Till Mossakowski. Foundations of heterogeneous specification. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *Recent Trends in Algebraic Development Techniques.. Selected Papers from the 16th International Workshop on Algebraic Development Techniques*, Frauenchiemsee, *Lecture Notes in Computer Science*, volume 2755, pages 359–375. Springer, 2003.

Mos04. Peter D. Mosses, editor. CASL *Reference Manual*. Number 2960 in Lecture Notes in Computer Science. Springer, 2004.

Mos05. Till Mossakowski. Heterogeneous Specification and the Heterogeneous Tool Set. Habilitation thesis, Universität Bremen, 2005.

MS85. David MacQueen and Donald Sannella. Completeness of proof systems for equational specifications. *IEEE Transactions on Software Engineering*, SE-11(5):454–461, 1985.

MSRR06. Till Mossakowski, Lutz Schröder, Markus Roggenbach, and Horst Reichel. Algebraic-coalgebraic specification in COCASL. *Journal of Logic and Algebraic Programming*, 67(1–2):146–197, 2006.

MSS90. Vincenzo Manca, Antonino Salibra, and Giuseppe Scollo. Equational type logic. *Theoretical Computer Science*, 77(1–2):131–159, 1990.

MST04. Till Mossakowski, Donald Sannella, and Andrzej Tarlecki. A simple refinement language for CASL. In José Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques.. Selected Papers from the 17th International Workshop on Algebraic Development Techniques*, Barcelona, *Lecture Notes in Computer Science*, volume 3423, pages 162–185. Springer, 2004.

MT92. Karl Meinke and John Tucker. Universal algebra. In Samson Abramsky, Dov Gabbay, and Tom Maibaum, editors, *Handbook of Logic in Computer Science. Volume 1 (Background: Mathematical Structures)*, pages 189–409. Oxford University Press, 1992.

MT93. V. Wiktor Marek and Mirosław Truszczyński. *Nonmonotonic Logics: Context-Dependent Reasoning*. Springer, 1993.

MT94. David B. MacQueen and Mads Tofte. A semantics for higher-order functors. In Donald Sannella, editor, *Proceedings of the 5th European Symposium on Programming*, Edinburgh, *Lecture Notes in Computer Science*, volume 788, pages 409–423. Springer, 1994.

MT09. Till Mossakowski and Andrzej Tarlecki. Heterogeneous logical environments for distributed specifications. In Andrea Corradini and Ugo Montanari, editors, *Recent Trends in Algebraic Development Techniques.. Selected Papers from the 19th International Workshop on Algebraic Development Techniques*, Pisa, *Lecture Notes in Computer Science*, volume 5486, pages 266–289. Springer, 2009.

MTD09. Till Mossakowski, Andrzej Tarlecki, and Răzvan Diaconescu. What is a logic translation? *Logica Universalis*, 3(1):95–124, 2009.

MTHM97. Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

MTP97. Till Mossakowski, Andrzej Tarlecki, and Wiesław Pawłowski. Combining and representing logical systems. In Eugenio Moggi and Giuseppe Rosolini, editors, *Proceedings of the 7th International Conference on Category Theory and Computer Science*,

Santa Margherita Ligure, *Lecture Notes in Computer Science*, volume 1290, pages 177–196. Springer, 1997.

MTP98. Till Mossakowski, Andrzej Tarlecki, and Wiesław Pawłowski. Combining and representing logical systems using model-theoretic parchments. In Francesco Parisi-Presicce, editor, *Recent Trends in Data Type Specification. Selected Papers from the 12th International Workshop on Specification of Abstract Data Types*, Tarquinia, *Lecture Notes in Computer Science*, volume 1376, pages 349–364. Springer, 1998.

MTW88. Bernhard Möller, Andrzej Tarlecki, and Martin Wirsing. Algebraic specifications of reachable higher-order algebras. In Donald Sannella and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification. Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, *Lecture Notes in Computer Science*, volume 332, pages 154–169. Springer, 1988.

Mus80. David Musser. On proving inductive properties of abstract data types. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, Las Vegas, pages 154–162, 1980.

MW98. Alfio Martini and Uwe Wolter. A single perspective on arrows between institutions. In Armando Haeberer, editor, *Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology*, Manaus, *Lecture Notes in Computer Science*, volume 1548, pages 486–501. Springer, 1998.

Nel91. Greg Nelson, editor. *Systems Programming in Modula-3*. Prentice-Hall, 1991.

Nip86. Tobias Nipkow. Non-deterministic data types: Models and implementations. *Acta Informatica*, 22(6):629–661, 1986.

NO88. Pilar Nivela and Fernando Orejas. Initial behaviour semantics for algebraic specifications. In Donald Sannella and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification. Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, *Lecture Notes in Computer Science*, volume 332, pages 184–207. Springer, 1988.

Nou81. Farshid Nourani. On induction for programming logic: Syntax, semantics, and inductive closure. *Bulletin of the European Association for Theoretical Computer Science*, 13:51–64, 1981.

Oka98. Chris Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.

ONS93. Fernando Orejas, Marisa Navarro, and Ana Sánchez. Implementation and behavioural equivalence: A survey. In Michel Bidoit and Christine Choppy, editors, *Recent Trends in Data Type Specification. Selected Papers from the 8th Workshop on Specification of Abstract Data Types joint with the 3rd* COMPASS *Workshop*, Dourdan, *Lecture Notes in Computer Science*, volume 655, pages 93–125. Springer, 1993.

Ore83. Fernando Orejas. Characterizing composability of abstract implementations. In Marek Karpinski, editor, *Proceedings of the 1983 International Conference on Foundations of Computation Theory*, Borgholm, *Lecture Notes in Computer Science*, volume 158, pages 335–346. Springer, 1983.

Pad85. Peter Padawitz. Parameter preserving data type specifications. In Hartmut Ehrig, Christiane Floyd, Maurice Nivat, and James Thatcher, editors, *TAPSOFT'85: Proceedings of the International Joint Conference on Theory and Practice of Software Development. Volume 2: Colloquium on Software Engineering*, Berlin, *Lecture Notes in Computer Science*, volume 186, pages 323–341. Springer, 1985.

Pad99. Peter Padawitz. Proof in flat specifications. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 10, pages 321–384. Springer, 1999.

Pau87. Laurence Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, 1987.

Pau96. Laurence Paulson. *ML for the Working Programmer*. Cambridge University Press, second edition, 1996.

Paw96. Wiesław Pawłowski. Context institutions. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification. Selected Papers from*

the 11th Workshop on Specification of Abstract Data Types, Oslo, *Lecture Notes in Computer Science*, volume 1130, pages 436–457. Springer, 1996.

Pet10. Marius Petria. *Generic Refinements for Behavioural Specifications*. PhD thesis, University of Edinburgh, School of Informatics, 2010.

Pey03. Simon Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

Pho92. Wesley Phoa. An introduction to fibrations, topos theory, the effective topos and modest sets. Technical Report ECS-LFCS-92-208, LFCS, Department of Computer Science, University of Edinburgh, 1992.

Pie91. Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, 1991.

Plo77. Gordon D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977.

Poi86. Axel Poigné. On specifications, theories, and models with higher types. *Information and Control*, 68(1–3):1–46, 1986.

Poi88. Axel Poigné. Foundations are rich institutions, but institutions are poor foundations. In Hartmut Ehrig, Horst Herrlich, Hans-Jörg Kreowski, and Gerhard Preuß, editors, *Proceedings of the International Workshop on Categorical Methods in Computer Science with Aspects from Topology*, Berlin, *Lecture Notes in Computer Science*, volume 393, pages 82–101. Springer, 1988.

Poi90. Axel Poigné. Parametrization for order-sorted algebraic specification. *Journal of Computer and System Sciences*, 40:229–268, 1990.

Poi92. Axel Poigné. Basic category theory. In Samson Abramsky, Dov Gabbay, and Tom Maibaum, editors, *Handbook of Logic in Computer Science. Volume 1 (Background: Mathematical Structures)*, pages 413–640. Oxford University Press, 1992.

Pos47. Emil Post. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 12:1–11, 1947.

PS83. Helmuth Partsch and Ralf Steinbrüggen. Program transformation systems. *ACM Computing Surveys*, 15(3):199–236, 1983.

PŞR09. Andrei Popescu, Traian Florin Şerbănuţă, and Grigore Roşu. A semantic approach to interpolation. *Theoretical Computer Science*, 410(12–13):1109–1128, 2009.

QG93. Xiaolei Qian and Allen Goldberg. Referential opacity in nondeterministic data refinement. *ACM Letters on Programming Languages and Systems*, 2(1–4):233–241, 1993.

Qia93. Zhenyu Qian. An algebraic semantics of higher-order types with subtypes. *Acta Informatica*, 30(6):569–607, 1993.

RAC99. Gianna Reggio, Egidio Astesiano, and Christine Choppy. CASL-LTL: a CASL extension for dynamic systems — summary. Technical Report DISI-TR-99-34, DISI, Università di Genova, 1999.

RB88. David Rydeheard and Rod Burstall. *Computational Category Theory*. Prentice Hall International Series in Computer Science. Prentice Hall, 1988.

Rei80. Horst Reichel. Initially-restricting algebraic theories. In Piotr Dembiński, editor, *Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science*, *Lecture Notes in Computer Science*, volume 88, pages 504–514, Rydzyna, 1980. Springer.

Rei81. Horst Reichel. Behavioural equivalence — a unifying concept for initial and final specification methods. In *Proceedings of the 3rd Hungarian Computer Science Conference*, pages 27–39, 1981.

Rei85. Horst Reichel. Behavioural validity of equations in abstract data types. In *Proceedings of the Vienna Conference on Contributions to General Algebra*, pages 301–324. Teubner-Verlag, 1985.

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

RG98. Grigore Roşu and Joseph A. Goguen. Hidden congruent deduction. In Ricardo Caferra and Gernot Salzer, editors, *Proceedings of the 1998 Workshop on First-Order Theorem Proving*, Vienna, *Lecture Notes in Artificial Intelligence*, volume 1761, pages 251–266. Springer, 1998.

RG00. Grigore Roşu and Joseph A. Goguen. On equational Craig interpolation. *Journal of Universal Computer Science*, 6(1):194–200, 2000.

Rod91. Pieter Hendrik Rodenburg. A simple algebraic proof of the equational interpolation theorem. *Algebra Universalis*, 28:48–51, 1991.

Rog06. Markus Roggenbach. CSP-CASL — a new integration of process algebra and algebraic specification. *Theoretical Computer Science*, 354(1):42–71, 2006.

Roş94. Grigore Roşu. The institution of order-sorted equational logic. *Bulletin of the European Association for Theoretical Computer Science*, 53:250–255, 1994.

Roş00. Grigore Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.

RRS00. Sergei Romanenko, Claudio Russo, and Peter Sestoft. Moscow ML owner's manual. Technical report, Royal Veterinary and Agricultural University, Copenhagen, 2000. Available from `http://www.itu.dk/people/sestoft/mosml/manual.pdf`.

RS63. Helena Rasiowa and Roman Sikorski. *The Mathematics of Metamathematics*. Number 41 in Monografie Matematyczne. Polish Scientific Publishers, 1963.

Rus98. Claudio Russo. *Types for Modules*. PhD thesis, University of Edinburgh, Department of Computer Science, 1998. Also in: *Electronic Notes in Theoretical Computer Science*, 60, 2003.

Rut00. Jan J.M.M. Rutten. Universal coalgebra: A theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

San82. Donald Sannella. *Semantics, Implementation and Pragmatics of Clear, a Program Specification Language*. PhD thesis, University of Edinburgh, Department of Computer Science, 1982.

SB83. Donald Sannella and Rod Burstall. Structured theories in LCF. In Giorgio Ausiello and Marco Protasi, editors, *Proceedings of the 8th Colloquium on Trees in Algebra and Programming*, L'Aquila, *Lecture Notes in Computer Science*, volume 159, pages 377–391. Springer, 1983.

Sch86. David Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, 1986.

Sch87. Oliver Schoett. *Data Abstraction and the Correctness of Modular Programs*. PhD thesis, University of Edinburgh, Department of Computer Science, 1987.

Sch90. Oliver Schoett. Behavioural correctness of data representations. *Science of Computer Programming*, 14(1):43–57, 1990.

Sch92. Oliver Schoett. Two impossibility theorems on behaviour specification of abstract data types. *Acta Informatica*, 29(6/7):595–621, 1992.

Sco76. Dana Scott. Data types as lattices. *SIAM Journal of Computing*, 5(3):522–587, 1976.

Sco04. Giuseppe Scollo. An institution isomorphism for planar graph colouring. In Rudolf Berghammer, Bernhard Möller, and Georg Struth, editors, *Relational and Kleene-Algebraic Methods in Computer Science. Selected Papers from the 7th International Seminar on Relational Methods in Computer Science and 2nd International Workshop on Applications of Kleene Algebra*, Bad Malente, *Lecture Notes in Computer Science*, volume 3051, pages 252–264. Springer, 2004.

SCS94. Amílcar Sernadas, José Félix Costa, and Cristina Sernadas. An institution of object behaviour. In Hartmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification. Selected Papers from the 9th Workshop on Specification of Abstract Data Types joint with the 4th* COMPASS *Workshop*, Caldes de Malavella, *Lecture Notes in Computer Science*, volume 785, pages 337–350. Springer, 1994.

Sel72. Alan Selman. Completeness of calculi for axiomatically defined classes of algebras. *Algebra Universalis*, 2:20–32, 1972.

SH00. Christopher A. Stone and Robert Harper. Deciding type equivalence in a language with singleton kinds. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, Boston, pages 214–227, 2000.

Sha08. Stewart Shapiro. Classical logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. CSLI, Stanford University, fall 2008 edition, 2008. Available from `http://plato.stanford.edu/archives/fall2008/entries/logic-classical/`.

SM09. Lutz Schröder and Till Mossakowski. HASCASL: Integrated higher-order specification and program development. *Theoretical Computer Science*, 410(12–13):1217–1260, 2009.

Smi93. Douglas R. Smith. Constructing specification morphisms. *Journal of Symbolic Computation*, 15(5/6):571–606, 1993.

Smi06. Douglas R. Smith. Composition by colimit and formal software development. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 4060, pages 317–332. Springer, 2006.

SML05. Lutz Schröder, Till Mossakowski, and Christoph Lüth. Type class polymorphism in an institutional framework. In José Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques.. Selected Papers from the 17th International Workshop on Algebraic Development Techniques*, Barcelona, *Lecture Notes in Computer Science*, volume 3423, pages 234–248. Springer, 2005.

Smo86. Gert Smolka. Order-sorted Horn logic: Semantics and deduction. Technical Report SR-86-17, Universität Kaiserslautern, Fachbereich Informatik, 1986.

SMT+05. Lutz Schröder, Till Mossakowski, Andrzej Tarlecki, Bartek Klin, and Piotr Hoffman. Amalgamation in the semantics of CASL. *Theoretical Computer Science*, 331(1):215–247, 2005.

Spi92. J. Michael Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, second edition, 1992.

SS93. Antonino Salibra and Guiseppe Scollo. A soft stairway to institutions. In Michel Bidoit and Christine Choppy, editors, *Recent Trends in Data Type Specification. Selected Papers from the 8th Workshop on Specification of Abstract Data Types joint with the 3rd* COMPASS *Workshop*, Dourdan, *Lecture Notes in Computer Science*, volume 655, pages 310–329. Springer, 1993.

SS96. Antonino Salibra and Giuseppe Scollo. Interpolation and compactness in categories of pre-institutions. *Mathematical Structures in Computer Science*, 6(3):261–286, 1996.

SST92. Donald Sannella, Stefan Sokołowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Parameterisation revisited. *Acta Informatica*, 29(8):689–736, 1992.

ST85. Donald Sannella and Andrzej Tarlecki. Program specification and development in Standard ML. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, New Orleans, pages 67–77, 1985.

ST86. Donald Sannella and Andrzej Tarlecki. Extended ML: An institution-independent framework for formal program development. In David H. Pitt, Samson Abramsky, Axel Poigné, and David E. Rydeheard, editors, *Proceedings of the Tutorial and Workshop on Category Theory and Computer Programming*, Guildford, *Lecture Notes in Computer Science*, volume 240, pages 364–389. Springer, 1986.

ST87. Donald Sannella and Andrzej Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Sciences*, 34:150–178, 1987.

ST88a. Donald Sannella and Andrzej Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76(2/3):165–210, 1988.

ST88b. Donald Sannella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica*, 25:233–281, 1988.

ST89. Donald Sannella and Andrzej Tarlecki. Toward formal development of ML programs: Foundations and methodology. In Josep Díaz and Fernando Orejas, editors, *TAPSOFT'89: Proceedings of the International Joint Conference on Theory and Practice of Software Development. Volume 2: Advanced Seminar on Foundations of Innovative Software Development II and Colloquium on Current Issues in Programming Languages*, Barcelona, *Lecture Notes in Computer Science*, volume 352, pages 375–389. Springer, 1989.

ST97. Donald Sannella and Andrzej Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, 9:229–269, 1997.

ST04. Donald Sannella and Andrzej Tarlecki, editors. CASL semantics. In *[Mos04]*. 2004.

ST06. Donald Sannella and Andrzej Tarlecki. Horizontal composability revisited. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning and Computation: Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 4060, pages 296–316. Springer, 2006.

ST08. Donald Sannella and Andrzej Tarlecki. Observability concepts in abstract data type specification, 30 years later. In Pierpaolo Degano, Rocco de Nicola, and José Meseguer, editors, *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of his 65th Birthday*, Lecture Notes in Computer Science. Springer, 2008.

Str67. Christopher Strachey. Fundamental concepts in programming languages. In *NATO Summer School in Programming, Copenhagen*. 1967. Also in: *Higher-Order and Symbolic Computation* 13(1–2):11–49, 2000.

SU06. Morten H. Sørensen and Paweł Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Number 149 in Studies in Logic and the Foundations of Mathematics. Elsevier Science, 2006.

SW82. Donald Sannella and Martin Wirsing. Implementation of parameterised specifications. In Mogens Nielsen and Erik Meineche Schmidt, editors, *Proceeding of the 9th International Colloquium on Automata, Languages and Programming*, Aarhus, *Lecture Notes in Computer Science*, volume 140, pages 473–488. Springer, 1982.

SW83. Donald Sannella and Martin Wirsing. A kernel language for algebraic specification and implementation. In Marek Karpinski, editor, *Proceedings of the 1983 International Conference on Foundations of Computation Theory*, Borgholm, *Lecture Notes in Computer Science*, volume 158, pages 413–427. Springer, 1983.

SW99. Donald Sannella and Martin Wirsing. Specification languages. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 8, pages 243–272. Springer, 1999.

Tar85. Andrzej Tarlecki. On the existence of free models in abstract algebraic institutions. *Theoretical Computer Science*, 37(3):269–304, 1985.

Tar86a. Andrzej Tarlecki. Bits and pieces of the theory of institutions. In David H. Pitt, Samson Abramsky, Axel Poigné, and David E. Rydeheard, editors, *Proceedings of the Tutorial and Workshop on Category Theory and Computer Programming*, Guildford, *Lecture Notes in Computer Science*, volume 240, pages 334–360. Springer, 1986.

Tar86b. Andrzej Tarlecki. Quasi-varieties in abstract algebraic institutions. *Journal of Computer and System Sciences*, 33(3):333–360, 1986.

Tar87. Andrzej Tarlecki. Institution representation. Unpublished note, Dept. of Computer Science, University of Edinburgh, 1987.

Tar96. Andrzej Tarlecki. Moving between logical systems. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification. Selected Papers from the 11th Workshop on Specification of Abstract Data Types*, Oslo, *Lecture Notes in Computer Science*, volume 1130, pages 478–502. Springer, 1996.

Tar99. Andrzej Tarlecki. Institutions: An abstract framework for formal specification. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 4, pages 105–130. Springer, 1999.

Tar00. Andrzej Tarlecki. Towards heterogeneous specifications. In Dov Gabbay and Maarten de Rijke, editors, *Frontiers of Combining Systems 2*, Studies in Logic and Computation, pages 337–360. Research Studies Press, 2000.

TBG91. Andrzej Tarlecki, Rod M. Burstall, and Joseph A. Goguen. Some fundamental algebraic tools for the semantics of computation. Part 3: Indexed categories. *Theoretical Computer Science*, 91(2):239–264, 1991.

Ter03. Terese. *Term Rewriting Systems*, *Cambridge Tracts in Theoretical Computer Science*, volume 55. Cambridge University Press, 2003.

Tho89. Simon Thompson. A logic for Miranda. *Formal Aspects of Computing*, 1(4):339–365, 1989.

TM87. Władysław M. Turski and Thomas S.E. Maibaum. *Specification of Computer Programs*. Addison-Wesley, 1987.

Tra93. Will Tracz. Parametrized programming in LILEANNA. In *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, Indianapolis, pages 77–86, 1993.

TWW82. James Thatcher, Eric Wagner, and Jesse Wright. Data type specification: Parameterization and the power of specification techniques. *ACM Transactions on Programming Languages and Systems*, 4(4):711–732, 1982.

Vra88. Jos L.M. Vrancken. The algebraic specification of semi-computable data types. In Donald Sannella and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification. Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, *Lecture Notes in Computer Science*, volume 332, pages 249–259. Springer, 1988.

Wad89. Philip Wadler. Theorems for free! In *Proceedings of the 4th International ACM Conference on Functional Programming Languages and Computer Architecture*, London, pages 347–359, 1989.

Wan79. Mitchell Wand. Final algebra semantics and data type extensions. *Journal of Computer and System Sciences*, 19:27–44, 1979.

Wan82. Mitchell Wand. Specifications, models, and implementations of data abstractions. *Theoretical Computer Science*, 20(1):3–32, 1982.

WB82. Martin Wirsing and Manfred Broy. An analysis of semantic models for algebraic specifications. In Manfred Broy and Gunther Schmidt, editors, *Theoretical Foundations of Programming Methodology: Lecture Notes of an International Summer School, Marktoberdorf 1981*, pages 351–416. Reidel, 1982.

WB89. Martin Wirsing and Manfred Broy. A modular framework for specification and implementation. In Josep Díaz and Fernando Orejas, editors, *TAPSOFT'89: Proceedings of the International Joint Conference on Theory and Practice of Software Development. Volume 1: Advanced Seminar on Foundations of Innovative Software Development I and Colloquium on Trees in Algebra and Programming*, Barcelona, *Lecture Notes in Computer Science*, volume 351, pages 42–73. Springer, 1989.

WE87. Eric G. Wagner and Hartmut Ehrig. Canonical constraints for parameterized data types. *Theoretical Computer Science*, 50:323–349, 1987.

Wec92. Wolfgang Wechler. *Universal Algebra for Computer Scientists*, *EATCS Monographs on Theoretical Computer Science*, volume 25. Springer, 1992.

Wik. Wikipedia. Hash table. Available from `http://en.wikipedia.org/wiki/Hash_table`.

Wir82. Martin Wirsing. Structured algebraic specifications. In *Proceedings of the AFCET Symposium on Mathematics for Computer Science*, Paris, pages 93–107, 1982.

Wir86. Martin Wirsing. Structured algebraic specifications: A kernel language. *Theoretical Computer Science*, 42(2):123–249, 1986.

Wir90. Martin Wirsing. Algebraic specification. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B (Formal Models and Semantics)*, pages 675–788. North-Holland and MIT Press, 1990.

Wir93. Martin Wirsing. Structured specifications: Syntax, semantics and proof calculus. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and*

Algebra of Specification: Proceedings of the NATO Advanced Study Institute, Marktoberdorf 1991, pages 411–442. Springer, 1993.

WM97. Michal Walicki and Sigurd Meldal. Algebraic approches to nondeterminism: An overview. *ACM Computing Surveys*, 29(1):30–81, 1997.

Zil74. Steven Zilles. Abstract specification of data types. Technical Report 119, Computation Structures Group, Massachusetts Institute of Technology, 1974.

Zuc99. Elena Zucca. From static to dynamic abstract data-types: An institution transformation. *Theoretical Computer Science*, 216(1–2):109–157, 1999.