1
Category Theory

Elementary category theory is concerned with categories, functors, and natural transformations. As described in Mac Lane (1998):

"category" has been defined in order to be able to define "functor" and "functor" has been defined in order to be able to define "natural transformation."

We shall consider each notion in turn, whilst simultaneously preparing the grounds for string diagrams to be introduced in Chapter 2.

1.1 Categories

A *category* consists of objects and arrows between objects. The letters $\mathcal{C}$, $\mathcal{D}$, … range over categories, and the uppercase letters A,B, … over objects. We write $A : \mathcal{C}$ to express that A is an object of the category $\mathcal{C}$. Lowercase letters f, g, … range over arrows, and we write $f : A \to B : \mathcal{C}$ to express that f is an arrow from A to B in the category $\mathcal{C}$. The object A is called the *source* of f and B its *target*. If $\mathcal{C}$ is obvious from the context, we abbreviate $f : A \to B : \mathcal{C}$ by $f : A \to B$.

For every object $A : \mathcal{C}$ there is an arrow $id_A : A \to A$, called the *identity*. Two arrows can be *composed* if their types match: if $f : A \to B$ and $g : B \to C$, then $g \cdot f : A \to C$ (pronounced "g after f"). We require composition to be unital and associative, with identity as its neutral element:

$$id_B \cdot f = f = f \cdot id_A, \tag{1.1a}$$

$$(h \cdot g) \cdot f = h \cdot (g \cdot f). \tag{1.1b}$$

1.1.1 Examples of Categories. To make the abstract notion of category more tangible, we introduce several examples, many of which will accompany us throughout the monograph. We begin with two trivial but useful categories:

Example 1.1 (**0** and **1**)**.** There is a category, denoted **0**, with no objects and no arrows. There is also a category **1**, with one object and one arrow, the identity on the single object. □

Categories can be seen as generalizations of possibly more familiar mathematical objects.

Example 1.2 (Monoids and Preorders)**.** Two extreme classes of categories are worth singling out.

A monoid $(A, e, \bullet)$ can be seen as a category that has exactly one object. The arrows are the elements of the monoid: e serves as the identity and $\bullet$ as composition.

A preorder $(A, \leqslant)$ can be seen as a category with at most one arrow between any two objects, which are the elements of the preorder. There exists an arrow of type $a \to b$ if and only if $a \leqslant b$; reflexivity ensures the existence of identities and transitivity the existence of composites. □

A category is often identified with its collection of objects: we loosely say that **Set** is the category of sets. However, equally if not more important are the arrows of a category. So, **Set** is really the category of sets and total functions. There is also **Rel**, the category of sets and relations.

Remark 1.3 (Preservation and Reflection of Structure)**.** An arrow*preserves* structure if features of the source allow us to deduce features of the target. For example, if $h : (A, 0, +) \to (B, 1, \times)$ is a monoid homomorphism, and $a + a' = 0$ holds in the source monoid, then $h\,a \times h\,a' = 1$ holds in the target monoid. This is exactly the motivation for homomorphisms between algebraic structures: they *preserve equations*.

An arrow *reflects* structure if we can infer properties of the source from properties of the target. Notice the backward direction of travel.

To illustrate this, let us first establish some useful notation that we need time and again. For a function $f : A \to B$ there is a *direct image function* taking subsets of A to subsets of B:

$$f^{\blacktriangleright}\, X := \{\, y \in B \mid \exists x \in X \,.\, f\,x = y \,\}.$$

There is also an *inverse image function*, mapping subsets in the opposite direction:

$$f^{\blacktriangleleft}\, Y := \{\, x \in A \mid \exists y \in Y \,.\, f\,x = y \,\}.$$

With this notation in place, if $h : A \to B$ is a continuous map of topological spaces, $Y \subseteq B$ being an open subset of B implies $f^{\blacktriangleleft}\, Y \subseteq A$ is an open set in A. So, structure in the target implies structure in the source, and these topological arrows reflect structure. □

Example 1.4 (Sets and Structures)**.** Many examples of categories used in practice are sets with additional structure, and functions that preserve or reflect this structure.

Sets with additional structure include monoids, groups, preorders, lattices, graphs, and so on. In each of these cases the arrows are structure-preserving maps. For example, **Mon** is the category of monoids and monoid homomorphisms. Notice the difference as compared to Example 1.2. There we considered a single monoid; here we consider the collection of all monoids and homomorphisms between them. Likewise, we can form the category **Pre**, whose objects are preorders and whose arrows are monotone or order-preserving functions.

Further examples include **Bool**, **Sup**, and **CompLat**, which are respectively the categories of Boolean lattices, complete join-semilattices, and complete lattices, with homomorphisms preserving the algebraic structure. Note that, although every complete join-semilattice is automatically a lattice, the categories **Sup** and **CompLat** are different, as the arrows preserve different structure.

As well as these examples with structure-preserving maps, there are examples where the arrows *reflect* structure, such as the categories **Top** and **Met** of topological spaces and metric spaces, with continuous maps as arrows. □

The following category, which will accompany us as a running example, is perhaps slightly more unusual.

Example 1.5 (Category of Actions)**.** Let $(M, e, \bullet)$ be a fixed monoid. The objects of the category M-**Act** are pairs $(A, \triangleleft)$, where A is a set and $(\triangleleft) : M \times A \to A$ is an operation that respects the monoid structure:

$$e \triangleleft a = a, \tag{1.2a}$$

$$(m \bullet n) \triangleleft a = m \triangleleft (n \triangleleft a). \tag{1.2b}$$

The operation is also called a *left action of* M. An arrow $f : (A, \triangleleft) \to (B, \blacktriangleleft)$ in M-**Act** is a function of type $A \to B$ that preserves actions:

$$f\,(m \triangleleft a) = m \blacktriangleleft f\,a, \tag{1.3}$$

also known as an *equivariant function*. □

There are many ways of constructing new categories from old, as we will see in later sections. For now, we consider three useful cases.

Definition 1.6 (Subcategories)**.** A *subcategory* $\mathcal{S}$ of a category $\mathcal{C}$ is a collection of some of the objects and some of the arrows of $\mathcal{C}$, such that identity and composition are preserved to ensure $\mathcal{S}$ constitutes a valid category. For example, **Set** is a subcategory of **Rel** as every function is a binary relation. Commutative monoids **CMon** and commutative, idempotent monoids **CIMon** form subcategories of **Mon**.

In a *full subcategory* the collection of arrows is maximal: if $f : A \to B : \mathcal{C}$ and $A, B : \mathcal{S}$, then $f : A \to B : \mathcal{S}$. The category **Fin** of finite sets and total functions is a full subcategory of **Set**. $\square$

Definition 1.7 (Opposite Categories)**.** For any category $\mathcal{C}$ we can consider its *opposite category* $\mathcal{C}^{op}$. This has the same objects as $\mathcal{C}$, but an arrow of type $A \to B$ in $\mathcal{C}^{op}$ is an arrow of type $B \to A$ in $\mathcal{C}$. Identities in $\mathcal{C}^{op}$ are as in $\mathcal{C}$, and composition in $\mathcal{C}^{op}$ is given by forming the reverse composite in $\mathcal{C}$. The process of swapping source and target is purely bureaucratic; it does not do anything to the arrows. $\square$

Definition 1.8 (Product Categories)**.** For any pair of categories $\mathcal{C}$ and $\mathcal{D}$ we can form their product $\mathcal{C} \times \mathcal{D}$. An object of the *product category* is a pair of objects (A, B) with $A : \mathcal{C}$ and $B : \mathcal{D}$; an arrow of type $(A, C) \to (B, D) : \mathcal{C} \times \mathcal{D}$ is a pair of arrows (f, g) with $f : A \to B : \mathcal{C}$ and $g : C \to D : \mathcal{D}$. Identity and composition are defined componentwise,

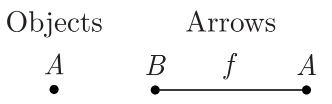
$$id_{(A,B)} := (id_A, id_B), \tag{1.4a}$$

$$(g_1, g_2) \cdot (f_1, f_2) := (g_1 \cdot f_1, g_2 \cdot f_2), \tag{1.4b}$$

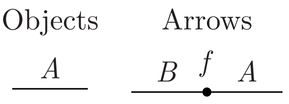
in terms of identity and composition of the underlying categories. $\square$

1.1.2 Graphical Representation of Objects and Arrows. We have noted in the prologue that notation matters, so a brief discussion of the syntax is certainly not amiss. Composition of arrows is a binary operation. Applications of binary operations or 2-ary functions are variably written prefix $op\,a\,b$, infix $a\,op\,b$, or postfix $a\,b\,op$, often with additional syntactic ornaments such as parentheses or commas. We have opted to write composition infix as $g \cdot f$. Why? Infix notation has a distinct advantage over the alternatives when expressions are nested as in $h \cdot g \cdot f$. At the outset, nested infix expressions are ambiguous, consider for example $a - b - c$. Do we intend to say $(a - b) - c$ or $a - (b - c)$? Convention has it that $a - b - c$ is resolved to $(a - b) - c$. For composition, however, the problem of ambiguity dissolves into thin air as composition is associative (1.1b). Here a bug has been turned into a feature: in calculations we do not have to invoke the associative law explicitly; it is built into the notation. By contrast, say we wrote composition prefix; then we are forced to express $h \cdot g \cdot f$ as either $\mathit{comp}\,(h, \mathit{comp}\,(g, f))$ or as $\mathit{comp}\,(\mathit{comp}\,(h, g), f)$. The syntax forces us to make an unwelcome distinction.

Composition of arrows in categories lends itself well to a graphical representation using vertices and edges. There are basically two options: objects can be represented by vertices, and arrows by edges between them, or vice versa:

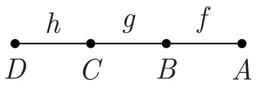


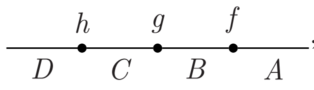
versus



.

The two types of diagrams are related by *topological* or *Poincaré duality*, where vertices become edges and edges become vertices. There are many variations of the two schemes. Vertices are often drawn as boxes or are not drawn at all, being replaced by their labels. Edges are often directed to allow for a more flexible arrangement of vertices. We avoid arrowheads by agreeing that the flow is from *right to left*. This choice blends well with the symbolic notation in that the graphical direction of composition,





,

follows the direction in the term $h \cdot g \cdot f$. For reasons of consistency, we should also write the types backwards: if $g : C \leftarrow B$ and $f : B \leftarrow A$, then $g \cdot f : C \leftarrow A$. We stick to the customary notation, however, and use right-to-left types only for emphasis. (An alternative is to change the order of composition: forward composition $f \mathbin{;} g \mathbin{;} h$ blends well with left-to-right types. We use both forward and backward composition.)

Like the symbolic notation, the diagrammatic representations have associativity (1.1b) built in, as we are simply threading beads on a necklace. We can further obviate the need for invoking unitality (1.1a) explicitly by agreeing that the identity arrow on an object A is represented by the rendition of A. The same convention is also used in symbolic notation: the identity on A is often written $A : A \to A$. A distinctive advantage of diagrams over terms is that they add vital type information. For a monoid $a \cdot b$ is always defined. However, as composition is in general partial, our notation should prevent us from joining arrows together incorrectly.

We have two graphical representations to choose from. But which one to pick? Different communities have different preferences: theoreticians seem to prefer the diagrams on the left above (e.g. as parts of commuting diagrams; see Section 1.7.2), while hardware people seem to prefer the diagrams on the right (e.g. in the form of circuit diagrams). We favor the latter notation for reasons that will become clear later.

1.2 Properties of Arrows

We now consider categorical generalizations of injective, surjective, and bijective functions.

1.2.1 Mono and Epi Arrows. An arrow $f : A \to B$ is called *mono* if it is *left-cancelable*:

$$f \cdot x_1 = f \cdot x_2 \implies x_1 = x_2, \tag{1.5a}$$

for all objects X and all arrows $x_1, x_2 : X \to A$. In **Set** these are the injective functions. Dually, an arrow $f : A \to B$ is called *epi* if it is *right-cancelable*:

$$x_1 \cdot f = x_2 \cdot f \implies x_1 = x_2, \tag{1.5b}$$

for all objects X and all arrows $x_1, x_2 : B \to X$. In **Set** these are the surjective functions. The inverse directions of the cancellation properties (1.5a) and (1.5b) are Leibniz's context rules,

$$x_1 = x_2 \implies f \cdot x_1 = f \cdot x_2, \tag{1.5c}$$

$$x_1 = x_2 \implies x_1 \cdot f = x_2 \cdot f, \tag{1.5d}$$

so implications (1.5a) and (1.5b) can both be strengthened to equivalences.

1.2.2 Split Mono and Split Epi Arrows. For an arrow $f : A \to B$, a *post-inverse* of f is an arrow $k : A \leftarrow B$ such that

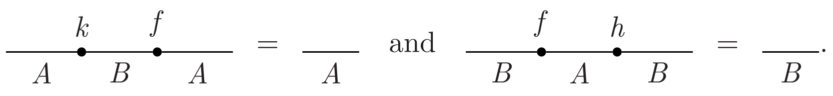
$$k \cdot f = id_A.$$

In this case, f is referred to as a *split mono*. Dually, a *pre-inverse* of f is an arrow $h : A \leftarrow B$ such that

$$f \cdot h = id_B.$$

Such an f is referred to as a *split epi*.

In pictures, these are arrows that annihilate each other if they touch in the right order:



Observe that the identity arrows are rendered by edges.

In **Set**, almost every injective function has a post-inverse. The only exceptions are functions of type $\emptyset \to A$ with $A \neq \emptyset$, simply because there are no functions of type $A \to \emptyset$. However, every surjective function has a pre-inverse.

Occasionally it is useful to reinterpret categorical notions using set-theoretic spectacles. If we partially apply the composition operator, $- \cdot f$ or $g \cdot -$, we obtain maps over collections of arrows. Using these maps we can reinterpret the notions of mono and epi. Property (1.5a) captures that $f \cdot -$ is injective; likewise, (1.5b) states that $- \cdot f$ is injective:

$$\begin{aligned} (f \cdot -)\, x_1 = (f \cdot -)\, x_2 \quad &\Longrightarrow \quad x_1 = x_2, \\ (- \cdot f)\, x_1 = (- \cdot f)\, x_2 \quad &\Longrightarrow \quad x_1 = x_2. \end{aligned}$$

While cancellation properties are related to injectivity, existence of a pre- or a post-inverse are related to surjectivity:

$$g \cdot - \text{ injective} \iff g \text{ mono}, \tag{1.6a}$$

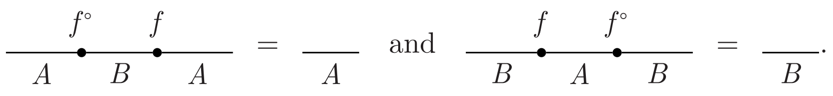
$$- \cdot f \text{ injective} \iff f \text{ epi}, \tag{1.6b}$$

$$g \cdot - \text{ surjective} \iff g \text{ split epi}, \tag{1.6c}$$

$$- \cdot f \text{ surjective} \iff f \text{ split mono}. \tag{1.6d}$$

The proofs of (1.6c) and (1.6d) are relegated to Exercise 1.8. The preceding list of equivalences partially explains why there are four different notions, rather than only two as in **Set**.

1.2.3 Isomorphisms. Two objects A and B are isomorphic, written $A \cong B$, if there is a pair of functions $f : A \to B$ and $g : A \leftarrow B$ such that $f \cdot g = id_B$ and $id_A = g \cdot f$. If an arrow $f : A \to B$ has both a pre- and a post-inverse, then they coincide, and we denote them f°. In this case f is an *isomorphism*, *iso* for short, with *inverse* f°, written $f : A \cong B : f^\circ$:



In **Set**, the isos are exactly the bijective functions.

The relation $\cong$ is an equivalence relation: it is reflexive, symmetric, and transitive. Furthermore, it is compatible with most constructions on objects. Reflexivity is established by identity arrows:

$$id_A : A \cong A : id_A.$$

Symmetry is shown by exchanging the isomorphisms:

$$f : A \cong B : f^\circ \quad \Longrightarrow \quad f^\circ : B \cong A : f.$$

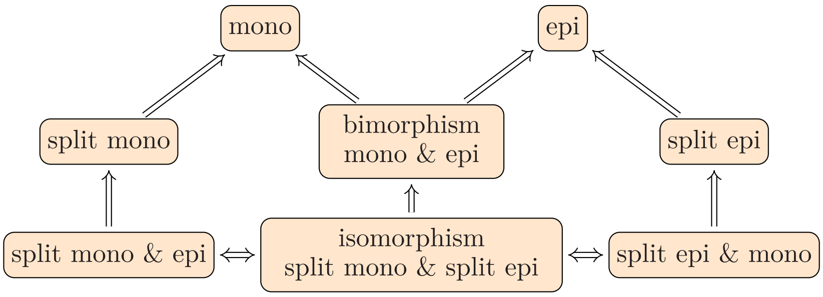
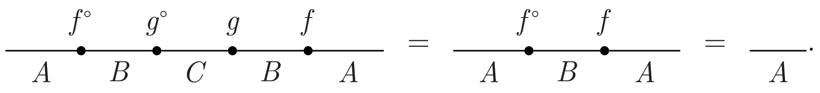


Figure 1.1 Properties of arrows.

Transitivity is established by suitably composing the witnesses:

$$f : A \cong B : f^\circ \quad \wedge \quad g : B \cong C : g^\circ \quad \Longrightarrow \quad g \cdot f : A \cong C : f^\circ \cdot g^\circ .$$

To prove that the composites are isomorphisms, we first annihilate the inner arrows and then the outer ones.



The proof for the reverse direction is entirely analogous.

Figure 1.1 relates the various properties of arrows – an *isomorphism* is an arrow that is both split mono and split epi; an arrow that is both mono and epi is called a *bimorphism*. The identity has all the properties, and all the properties are preserved by composition. Exercise 1.11 asks you to establish the relations and to show that the inclusions are proper.

The attentive reader may have noted that categorical concepts come in pairs. An epi in $\mathcal{C}$ is a mono in $\mathcal{C}^{\mathsf{op}}$; a split epi in $\mathcal{C}$ is a split mono in $\mathcal{C}^{\mathsf{op}}$; the concept of an iso is self-dual; an iso in $\mathcal{C}$ is an iso in $\mathcal{C}^{\mathsf{op}}$. Duality means that we get two concepts for the price of one. The next section provides further evidence for the economy of expression afforded by duality.

1.3 Thinking in Terms of Arrows

A category consists of objects and arrows. However, these entities are not treated on an equal footing: category theory puts the conceptual emphasis on arrows. Indeed, to master the subject one has to learn to think in terms of arrows. To illustrate, let us define some additional infrastructure: initial and final objects, products and coproducts, and exponentials. In each case, the defined object is characterized in terms of its relationship to other objects. In a sense, category theory is the most social of all mathematical foundations.

1.3.1 Initial and Final Objects. Let $\mathcal{C}$ be a category. An object $0 : \mathcal{C}$ is called *initial in* $\mathcal{C}$ if, for each object $A : \mathcal{C}$, there is exactly one arrow from the initial object 0 to A. This property is referred to as the *universal property* of initial objects.

Dually, $1 : \mathcal{C}$ is a *final* or *terminal object in* $\mathcal{C}$ if it satisfies the *universal property* that, for each object $A : \mathcal{C}$, there is a unique arrow from A to 1. A final object in $\mathcal{C}$ is an initial object in $\mathcal{C}^{\mathsf{op}}$.

An object that is simultaneously initial and final in $\mathcal{C}$ is called a *zero object*.

Example 1.9 (Preorders)**.** An initial object in a preorder category is a least element. Dually, a final object is a greatest element. If the preorder is a partial order, meaning the relation $\leqslant$ is also antisymmetric, then initial and final objects are unique. □

Example 1.10 (Sets and Structures)**.** In **Set**, the empty set is initial and *any* singleton set is final. In **Mon**, the singleton monoid $(\{()\}, (), \bullet)$ with $() \bullet () = ()$ is both initial and final, as homomorphisms have to preserve the neutral element: the singleton monoid is a zero object. □

The examples demonstrate that, in general, initial and final objects are not unique. They are, however, *unique up to a unique isomorphism*. If A and B are both initial, then there are unique arrows of type $A \to B$ and $B \to A$, whose compositions are necessarily identities.

1.3.2 Products and Coproducts. A *product* of two objects B_1 and B_2 consists of an object written as $B_1 \times B_2$ and a pair of *projection arrows*:

$$\mathit{outl} : B_1 \times B_2 \to B_1 \quad \text{and} \quad \mathit{outr} : B_1 \times B_2 \to B_2.$$

These three entities have to satisfy the following *universal property*: for each object A and for each pair of arrows $f_1 : A \to B_1$ and $f_2 : A \to B_2$, there exists an arrow $f_1 \vartriangle f_2 : A \to B_1 \times B_2$ (pronounced "f_1 *split* f_2") such that

$$f_1 = \mathit{outl} \cdot g \;\wedge\; f_2 = \mathit{outr} \cdot g \iff f_1 \vartriangle f_2 = g, \tag{1.7}$$

for all $g : A \to B_1 \times B_2$. The equivalence captures the existence of an arrow satisfying the property on the left and furthermore states that $f_1 \vartriangle f_2$ is the *unique* such arrow. The following commutative diagram (see Section 1.7.2) summarizes the type information:

$$\begin{tikzcd}
B_1 & & \\
& B_1 \times B_2 \arrow[ul, "outl"] \arrow[dl, "outr"'] & A \arrow[l, dotted, "f_1 \vartriangle f_2"'] \arrow[ull, bend right, "f_1"'] \arrow[dll, bend left, "f_2"] \; . \\
B_2 & &
\end{tikzcd}$$

The dotted arrow indicates that $f_1 \vartriangle f_2$ is the unique arrow from A to $B_1 \times B_2$ that makes the diagram commute. Informally, the universal property states that for anything that "looks like" a product there is a unique arrow that factorizes the look-alike product in terms of a "real" product. Section 5.2 makes the notion of a universal construction precise.

The construction of products dualizes to coproducts, which are products in the opposite category. The *coproduct* of two objects A_1 and A_2 consists of an object written as $A_1 + A_2$ and a pair of *injection arrows*:

$$inl : A_1 \to A_1 + A_2 \quad \text{and} \quad inr : A_2 \to A_1 + A_2 .$$

These three entities have to satisfy the following *universal property*: for each object B and for each pair of arrows $g_1 : A_1 \to B$ and $g_2 : A_2 \to B$, there exists an arrow $g_1 \triangledown g_2 : A_1 + A_2 \to B$ (pronounced "g_1 *join* g_2") such that

$$f = g_1 \triangledown g_2 \iff f \cdot inl = g_1 \;\wedge\; f \cdot inr = g_2, \tag{1.8}$$

for all $f : A_1 + A_2 \to B$. Reversing the arrows in the previous product diagram, we obtain the corresponding diagram for coproducts:

$$\begin{tikzcd}
A_1 \arrow[dr, "inl"'] \arrow[drr, bend left, "g_1"] & & \\
& A_1 + A_2 \arrow[r, dotted, "g_1 \triangledown g_2"] & B \; . \\
A_2 \arrow[ur, "inr"] \arrow[urr, bend right, "g_2"'] & &
\end{tikzcd}$$

Remark 1.11 (Bigger Products and Coproducts)**.** We have introduced binary products. Clearly, we can also define ternary products, with three projection arrows, and a corresponding universal property. These can be built by nesting binary products, with the order of composition unimportant, as the two choices are isomorphic:

$$(X_1 \times X_2) \times X_3 \cong X_1 \times (X_2 \times X_3).$$

This game can be repeated to form the product of any finite collection of objects. As the notation will quickly become clumsy, we write the *indexed product* of an I-indexed family of objects $X_{i \in I}$ as $\prod_{i \in I} X_i$.

It is then natural to broaden things further, considering *infinite products*, where i ranges over an infinite set, although these cannot be formed by iterating the finite construction, and so a category with finite products may not have these larger ones. In general, we have a set of projection arrows, indexed by i, and for every family of arrows $f_i : A \to X_i$ a unique mediating arrow of type $A \to \prod_{i \in I} X_i$, generalizing $f_1 \vartriangle f_2$.

Dually, we can form *iterated*, potentially *infinite* coproducts, over some index set I; these will be denoted $\sum_{i \in I} X_i$. □

Example 1.12 (Preorders). In preorder categories, products are greatest lower bounds or meets, and coproducts are least upper bounds or joins. The *meet* of b_1 and b_2, written as $b_1 \sqcap b_2$, is defined by the equivalence

$$a \leqslant b_1 \;\wedge\; a \leqslant b_2 \iff a \leqslant b_1 \sqcap b_2. \tag{1.9a}$$

Dually, the *join* of a_1 and a_2, written $a_1 \sqcup a_2$, is defined by

$$a_1 \sqcup a_2 \leqslant b \iff a_1 \leqslant b \;\wedge\; a_2 \leqslant b. \tag{1.9b}$$

The uniqueness conditions are trivially satisfied as there is at most one arrow between any two objects. □

Example 1.13 (Sets). In the category of sets and total functions, the product is given by the Cartesian product:

$$B_1 \times B_2 := \{(b_1, b_2) \mid b_1 \in B_1, b_2 \in B_2\}.$$

The split operator and the projection functions are defined by

$$(f_1 \vartriangle f_2)\, a := (f_1\, a, f_2\, a) \qquad \text{and} \qquad \begin{aligned} \mathit{outl}\,(b_1, b_2) &:= b_1, \\ \mathit{outr}\,(b_1, b_2) &:= b_2. \end{aligned}$$

The coproduct corresponds to the *disjoint* union of sets:

$$A_1 + A_2 := \{(0, a_1) \mid a_1 \in A_1\} \cup \{(1, a_2) \mid a_2 \in A_2\}.$$

The injection functions and the join operator are defined by

$$\begin{aligned} \mathit{inl}\; a_1 &:= (0, a_1) \\ \mathit{inr}\; a_2 &:= (1, a_2) \end{aligned} \qquad \text{and} \qquad \begin{aligned} (g_1 \triangledown g_2)\,(0, a_1) &:= g_1\, a_1, \\ (g_1 \triangledown g_2)\,(1, a_2) &:= g_2\, a_2. \end{aligned}$$

The injection functions "tag" their arguments; the join operator performs a case analysis on the tag.

The category **Rel** has the same coproducts as **Set**. However, the products differ: as **Rel** is self-dual, coproducts and products coincide. □

Observe that the duality of the "interfaces" – the universal properties (1.7) and (1.8) are like abstract interfaces – is not reflected in the "implementations" – the concrete definitions of the functions in **Set**. This leads to a second observation. A proof about products that is conducted in terms of the interface dualizes effortlessly to a proof about coproducts. A proof that is couched in terms of the concrete implementation is unlikely to enjoy the same reuse.

Like initial and final objects, products and coproducts are only defined up to isomorphism. (This statement can be sharpened, see Exercise 1.14.)

1.3.3 Exponentials. The *exponential* of two objects X and B in a category with products consists of an object written B^X and an arrow $apply : B^X \times X \to B$. These two entities have to satisfy the following *universal property*: for each object A and for each arrow $f : A \times X \to B$ there exists an arrow $\Lambda f : A \to B^X$ (pronounced "*curry f*") such that

$$f = apply \cdot (g \times id_X) \iff \Lambda f = g, \tag{1.10}$$

for all $g : A \to B^X$. The product of arrows is defined $h \times k := (h \cdot outl) \vartriangle (k \cdot outr)$, see also Example 1.19. Like for products, the equivalence captures the existence of an arrow satisfying the property on the left and furthermore states that Λf is the *unique* such arrow, as shown in the following diagram:

$$\begin{tikzcd}
B^X \times X \arrow[dr, "apply"'] & & A \times X \arrow[ll, dotted, "(\Lambda f)\times id_X"'] \arrow[dl, "f"] \\
& B &
\end{tikzcd} .$$

Example 1.14 (Preorders)**.** In a Boolean lattice the exponential b^x is given by $b \sqcup \neg x$. It satisfies the following equivalence:

$$a \sqcap x \leqslant b \iff a \leqslant b \sqcup \neg x.$$

In the smallest, nontrivial Boolean lattice $\mathbb{B}$, the exponential is often written as an implication: $b \Leftarrow x := b \sqcup \neg x$. □

Example 1.15 (Sets)**.** In **Set**, the exponential B^X amounts to the set of total functions from X to B. The operation Λ turns a two-argument function into a so-called curried function, a single-argument function of the first parameter that yields another single-argument function, which receives the second parameter. □

1.4 Functors

Every worthwhile algebraic *structure* comes equipped with corresponding *structure*-preserving maps; so do categories, where these maps are called *functors*. We let F, G, … range over functors. Since a category consists of two parts, objects and arrows, a functor $\mathsf{F} : \mathcal{C} \to \mathcal{D}$ consists of a mapping on objects and a mapping on arrows, called the *object map* and the *arrow map* respectively. It is common practice to denote both maps by the same symbol. The two parts of a functor have to be consistent; for one thing, the action on arrows has to respect the types: if $f : A \to B : \mathcal{C}$, then $\mathsf{F}\,f : \mathsf{F}\,A \to \mathsf{F}\,B : \mathcal{D}$. Furthermore, a functor has to preserve identities and composition:

$$\mathsf{F}\,(id_A) = id_{\mathsf{F}\,A}, \tag{1.11a}$$

$$\mathsf{F}\,(g \cdot f) = \mathsf{F}\,g \cdot \mathsf{F}\,f. \tag{1.11b}$$

These are called the *functor laws*. Equation (1.11a) is the *functor identity law*, and Equation (1.11b) is the *functor composition law*.

A simple example of a functor is the *identity* $\mathsf{Id}_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}$, defined as

$$\mathsf{Id}_{\mathcal{C}}\,A := A,$$
$$\mathsf{Id}_{\mathcal{C}}\,f := f.$$

Functoriality is preserved under composition: given two functors $\mathsf{F} : \mathcal{C} \to \mathcal{D}$ and $\mathsf{G} : \mathcal{D} \to \mathcal{E}$, their *composite* $\mathsf{G}{\circ}\mathsf{F} : \mathcal{C} \to \mathcal{E}$ is defined as

$$(\mathsf{G}{\circ}\mathsf{F})\,A := \mathsf{G}\,(\mathsf{F}\,A),$$
$$(\mathsf{G}{\circ}\mathsf{F})\,f := \mathsf{G}\,(\mathsf{F}\,f).$$

Categories and functors between them themselves form a category, called **Cat**. To avoid paradoxes similar to Russell's paradox, this construction is subject to size constraints, see I.6 of Mac Lane (1998).

We will revisit functor composition in Section 2.1.

1.4.1 Examples of Functors.

As we did with categories, we now consider some examples of functors to make things more concrete.

Example 1.16 (Functors between Monoids and Preorders)**.** We saw in Example 1.2 that monoids and preorders can be seen as categories. A functor between monoid categories is a monoid homomorphism; a functor between preorder categories is a monotone function. So, in this case, functors are the usual notion of homomorphism. □

Example 1.17 (Functors from **0**, and to **1**)**.** In Example 1.1 we introduced the empty category **0**. This category has the special property that there is exactly one functor to any category from **0**, as there is no data to specify. Similarly, the one object category **1** has the special property that there is exactly one functor from any category to **1**, as there is only one choice for where to send every object and arrow. In other words, **0** is the initial object in **Cat** and **1** is the final object. □

Example 1.18 (Functors from and to Product Categories)**.** A product category comes equipped with two projection functors, $\mathsf{Outl} : \mathcal{C}_1 \times \mathcal{C}_2 \to \mathcal{C}_1$ and $\mathsf{Outr} : \mathcal{C}_1 \times \mathcal{C}_2 \to \mathcal{C}_2$, defined as $\mathsf{Outl}\,(A_1, A_2) := A_1$, $\mathsf{Outl}\,(f_1, f_2) := f_1$ and $\mathsf{Outr}\,(A_1, A_2) := A_2$, $\mathsf{Outr}\,(f_1, f_2) := f_2$. A functor from a product category such as Outl and Outr is sometimes called a *bifunctor*, a contraction of the more unwieldy term *binary functor* (see also Exercise 1.23).

The *diagonal functor* $\Delta : \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ is an example of a functor into a product category. It duplicates its argument:

$$\Delta\, A := (A, A), \tag{1.12a}$$

$$\Delta\, f := (f, f). \tag{1.12b}$$

We have $\mathsf{Outl} \circ \Delta = \mathsf{Id}_{\mathcal{C}_1}$ and $\mathsf{Outr} \circ \Delta = \mathsf{Id}_{\mathcal{C}_2}$. □

Example 1.19 (Products and Exponentials)**.** If the product $B_1 \times B_2$ exists for every pair of objects, $- \times =$ can be turned into a bifunctor with arrow map:

$$g_1 \times g_2 = (g_1 \cdot \mathit{outl}) \vartriangle (g_2 \cdot \mathit{outr}).$$

Likewise, if the exponential B^X exists for every object B, $(-)^X$ can be turned into a functor with arrow map:

$$f^X = \Lambda\,(f \cdot \mathit{apply}).$$

Exercise 1.24 asks you to fill in the details. □

Many common mathematical constructions are functorial.

Example 1.20 (Powerset)**.** Given a set A, we can form its powerset $\mathcal{P}\,A$ consisting of all subsets of A. This extends to a functor $\mathsf{Pow} : \mathbf{Set} \to \mathbf{Set}$, referred to as the *covariant powerset functor*, with action on arrows:

$$\mathsf{Pow}\, f := f^{\blacktriangleright}.$$

In fact, still taking the powerset construction on objects, we can extend this in a second way to a functor $2^{(-)} : \mathbf{Set}^{op} \to \mathbf{Set}$:

$$2^f := f^{\blacktriangleleft}.$$

This is referred to as the *contravariant powerset functor*, as it reverses the direction of arrows: $2^{g \cdot f} = 2^f \cdot 2^g$. The notation is inspired by the notation for exponentials: if we represent a set $X : \mathcal{P}\, A$ by its characteristic function $\chi : A \to 2$, then the action on arrows reads $2^f\, \psi = \psi \cdot f$, and consequently $2^f\,(2^g\, \psi) = 2^f\,(\psi \cdot g) = \psi \cdot g \cdot f = 2^{g \cdot f}\, \psi$. □

The relationships between categories are important, and functors allow us to describe these relationships by showing how one can be transformed into the other.

Example 1.21 (Algebra and Order Theory)**.** A bounded join-semilattice is a partially ordered set with a bottom element, denoted $\bot$, and such that every pair of elements x, y has a least upper bound, or join, denoted $x \sqcup y$ (see Example 1.12).

Every commutative, idempotent monoid gives rise to a bounded join-semilattice, with order defined by $x \leqslant y \iff x \bullet y = y$. The construction is functorial, as we can define:

$$\begin{aligned} \mathsf{Ord}\,(A, e, \bullet) &:= (A, \leqslant) \quad \textbf{where} \quad x \leqslant y \iff x \bullet y = y, \\ \mathsf{Ord}\, h &:= h. \end{aligned}$$

Conversely, given a bounded join-semilattice, we can define a commutative, idempotent monoid functorially, with the monoid operations given by the bottom element and joins:

$$\begin{aligned} \mathsf{CIMon}\,(A, \leqslant) &:= (A, \bot, \sqcup), \\ \mathsf{CIMon}\, h &:= h. \end{aligned}$$

In fact, $\mathsf{Ord} \circ \mathsf{CIMon} = \mathsf{Id}$ and $\mathsf{CIMon} \circ \mathsf{Ord} = \mathsf{Id}$, so these constructions are inverse to each other, and we say that the categories of commutative, idempotent monoids and bounded join-semilattices are isomorphic. □

For many functors there are often natural ways to travel back in the opposite direction. We return to monoids for an instructive example.

Example 1.22 (Free and Forgetful)**.** The category **Mon** is based on the category **Set** by adding more structure. This informal statement can be made precise via a functor. The *underlying* or *forgetful functor* $\mathsf{U} : \mathbf{Mon} \to$

Set is defined as

$$\begin{aligned} \mathsf{U}\,(A, e, \bullet) &:= A, \\ \mathsf{U}\,h &:= h. \end{aligned} \tag{1.13}$$

The definition uses the fact that arrows in **Mon** are total functions. Since the action on arrows is the identity, the functoriality requirements are trivially satisfied.

The functor U has a counterpart, which takes an arbitrary set to the free monoid on the set. For that reason it is called the *free functor* $\mathsf{Free} : \mathbf{Set} \to \mathbf{Mon}$ and is defined as

$$\mathsf{Free}\,A := (A^*, [\,], +\!\!+),$$

$$\mathsf{Free}\,f := h\,\mathbf{where} \begin{cases} h\,[\,] := [\,], \\ h\,[a] := [f\,a], \\ h\,(x +\!\!+ y) := h\,x +\!\!+ h\,y. \end{cases}$$

Here, A^* is the set of all finite lists, whose elements are drawn from A. Lists can be constructed in three different ways: $[\,]$ is notation for the empty list, $[a]$ for the singleton list containing a, and $+\!\!+$ concatenates two lists. Concatenation is associative with $[\,]$ as its neutral element, so $(A^*, [\,], +\!\!+)$ is indeed a monoid. Furthermore, $\mathsf{Free}\,f$ is a monoid homomorphism by definition. It applies f to every element of a given list.

Composing the two functors gives an endofunctor on **Set**, which we call $\mathsf{List} := \mathsf{U} \circ \mathsf{Free} : \mathbf{Set} \to \mathbf{Set}$. The prefix "endo" emphasizes that source and target category of the functor are identical. □

Example 1.23 (Polynomial Functors)**.** A useful class of functors that commonly occur in practice are the *polynomial functors* on **Set**. These are functors built out of the iterated coproducts described in Remark 1.11, and exponentials. A polynomial functor is a functor of the following form:

$$X \;\mapsto\; \sum_{i \in I} X^{A_i}.$$

Here, the A_i are fixed sets. If we think of coproducts and exponentials as adding and raising to powers, the analogy with polynomials is clear.

Clearly the identity functor, $X \mapsto X^2$, and $X \mapsto X + X$ are examples of polynomial functors. Perhaps more surprisingly, the functor List of Example 1.22 is isomorphic to the polynomial functor:

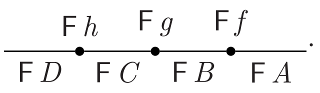
$$X \;\mapsto\; \sum_{n \in \mathbb{N}} X^n.$$

As X^n is a tuple of n elements, our polynomial functor contains tuples of all potential lengths, exactly capturing lists of elements of X. $\square$

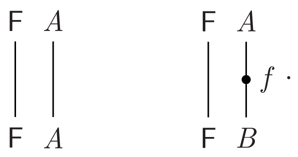
One of the great strengths of category theory is its ability to build bridges between different areas of mathematics and computer science. The following miniature example points in that direction.

Example 1.24 (Category of Actions)**.** Recall that a monoid $(M, e, \bullet)$ can be seen as a category $\mathcal{M}$. A functor $\mathcal{M} \to \mathbf{Set}$ is morally the same as an action of M; see Example 1.5. Since a monoid category has exactly one object, F selects a set and sends the monoid elements to endofunctions over that set. The coherence conditions for actions (1.2a) and (1.2b) correspond to the two functor laws. $\square$

1.4.2 Graphical Representation of Functors. Let us extend the graphical representation of objects and arrows to functors. The coherence of the object and arrow maps and the functor laws allow us to push applications of functors inwards. Thus, a simple but slightly unimaginative approach is to label diagrams with functor applications:

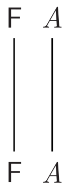


One could argue that the use of complex terms as labels mixes symbolic and diagrammatic notation. A more attractive alternative is to draw the functor as a separate wire extending diagrams to two dimensions. The application of a functor F to an object A and to an arrow f is rendered:

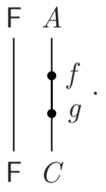


For reasons to become clear in Chapter 2, we have additionally rotated the diagrams 90° counterclockwise. We have also moved the labels to the ends of the wires to avoid cluttering the middle of the diagram.

Quite pleasingly, the functor laws are built into the notation, in the sense that equal terms have the same diagrams. Both sides of the identity functor law (1.11a) have the same depiction (1.14a). Similarly, both sides of the functor composition law (1.11b) correspond to diagram (1.14b):

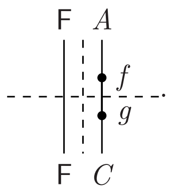


(1.14a)



(1.14b)

Diagram (1.14b) can be divided into four parts as follows:



The functor composition law (1.11b) implies that it does not matter whether we assemble the parts first horizontally and then vertically, or vice versa.

1.5 Natural Transformations

Category theorists never study objects in isolation; they always consider what the right arrows between those objects should be. To this end, in Section 1.4 we introduced functors as the arrows between categories. As we have now introduced functors, we can play this game again. We should ask ourselves, what should the arrows between functors be? Answering this question leads to the definition of *natural transformations*. Before getting into the formal bureaucracy, we start with an example.

Consider the function that maps an element to a singleton list, $a \mapsto [\,a\,]$, and observe that the definition does not depend in any way on the nature of the element a. In particular, the map works uniformly across all possible *element types*. This characteristic can be made precise using the notion of a natural transformation.

Let $\mathsf{F}, \mathsf{G} : \mathcal{C} \to \mathcal{D}$ be two functors. A *transformation* $\alpha : \mathsf{F} \to \mathsf{G} : \mathcal{C} \to \mathcal{D}$ is a family of arrows, so that for each object $A : \mathcal{C}$ there is an arrow $\alpha\, A : \mathsf{F}\, A \to \mathsf{G}\, A : \mathcal{D}$. The arrow $\alpha\, A$ is called a *component* of α. A transformation can be seen as a map from objects to arrows.

A transformation is *natural*, written $\alpha : \mathsf{F} \mathrel{\dot{\to}} \mathsf{G} : \mathcal{C} \to \mathcal{D}$, if

$$\mathsf{G}\, h \cdot \alpha\, X = \alpha\, Y \cdot \mathsf{F}\, h, \tag{1.15}$$

for all arrows $h : X \to Y : \mathcal{C}$. Given α and h, there are essentially two ways of turning $\mathsf{F}\, X$ entities into $\mathsf{G}\, Y$ entities. The *naturality condition* demands that they are equal. We let α, β, ... range over natural transformations.

Condition (1.15) is equivalent to requiring commutativity of the following diagram:

$$\begin{tikzcd}
\mathsf{G}\,X \arrow[d, "\mathsf{G}\,h"'] & \mathsf{F}\,X \arrow[l, "\alpha\,X"'] \arrow[d, "\mathsf{F}\,h\,\cdot"] \\
\mathsf{G}\,Y & \mathsf{F}\,Y \arrow[l, "\alpha\,Y"]
\end{tikzcd}$$

Such a diagram is termed a *naturality square*.

As always, it is important to consider identities and composition. For any functor $\mathsf{F} : \mathcal{C} \to \mathcal{D}$ the identity arrows $id_{\mathsf{F}\,X} : \mathsf{F}\,X \to \mathsf{F}\,X$ form an *identity natural transformation* of type $\mathsf{F} \dot{\to} \mathsf{F}$,

$$id_{\mathsf{F}}\,X := id_{\mathsf{F}\,X},$$

with the naturality condition becoming trivial. Given three parallel functors $\mathsf{F}, \mathsf{G}, \mathsf{H} : \mathcal{C} \to \mathcal{D}$ and natural transformations $\alpha : \mathsf{F} \dot{\to} \mathsf{G}$ and $\beta : \mathsf{G} \dot{\to} \mathsf{H}$, we can form their *vertical composite* $\beta \cdot \alpha$ with components

$$(\beta \cdot \alpha)\,X := \beta\,X \cdot \alpha\,X.$$

The naturality condition follows immediately from the naturality of the two components. For fixed categories $\mathcal{C}$ and $\mathcal{D}$, functors of type $\mathcal{C} \to \mathcal{D}$ and natural transformations between these functors form a category, the *functor category* $\mathcal{D}^{\mathcal{C}}$.

Remark 1.25. We have seen product categories in Definition 1.8. In fact, they are the categorical products in **Cat**, and functor categories are the corresponding exponentials. This is a way of seeing that natural transformations are the right choice of arrows between functors. □

We will revisit composition of natural transformations in Section 2.2. In particular, we shall see that there are also horizontal composites, and that the terms "vertical" and "horizontal" correspond to the graphical depiction of these different forms of composition.

1.5.1 Examples of Natural Transformations. As with the other members of the trinity of categorical concepts, we consider explicit examples of natural transformations. We begin with our usual favorites.

Example 1.26 (Monoid and Preorder Naturality)**.** For monoids, a natural transformation between monoid homomorphisms $f, g : X \to Y$ is an element α of Y such that $g\,x \cdot \alpha = \alpha \cdot f\,x$. That is, the two homomorphisms are related pointwise by conjugation by the element α.

We saw that functors between preorders are monotone functions. In this special case there can be at most one natural transformation between such functors. In fact, for $f, g : X \to Y$ there is a natural transformation $f \mathrel{\dot{\to}} g$ if and only if $f\,x \leqslant g\,x$ for all $x \in X$. That is, if f is below g pointwise: $f \leqslant g$. The naturality condition is trivially satisfied as there is at most one arrow between any two objects. □

When categories are specialized to preorders, coherence conditions such as (1.15) are often vacuous. Turning things around, category theory can be seen as *order theory with coherence conditions* (Backhouse et al., 1998). An "order" between two objects, say A and B, is *witnessed* by one or more arrows of type $A \to B$. The coherence conditions ensure that the choice of the witness is compatible with the basic operations of category theory, identifying witnesses if necessary.

We saw in Example 1.20 that the powerset construction is functorial. Many operations on subsets are natural.

Example 1.27 (Operations on Sets)**.** The singleton set functions,

$$\mathit{single}\;A\;a := \{\,a\,\},$$

form a natural transformation $\mathit{single} : \mathsf{Id} \mathrel{\dot{\to}} \mathsf{Pow}$. The naturality condition $\mathsf{Pow}\,f \cdot \mathit{single}\;A = \mathit{single}\;B \cdot f$ holds by definition. Similarly, taking unions gives a natural transformation $\bigcup : \mathsf{Pow} \circ \mathsf{Pow} \mathrel{\dot{\to}} \mathsf{Pow}$. □

Many common computational tasks are natural.

Example 1.28 (Reducing Lists)**.** For each monoid $(A, e, \bullet)$ there is a monoid homomorphism, which reduces a list to a single element of A:

$$\mathit{reduce}\,(A, e, \bullet) := h \;\mathbf{where}\; \begin{cases} \quad\; h\,[\,] := e, \\ \quad h\,[\,a\,] := a, \\ h\,(x \mathbin{+\!\!+} y) := h\,x \bullet h\,y. \end{cases}$$

Instantiating the monoid to $(\mathbb{N}, 0, +)$, it sums a list of natural numbers. For $(\mathbb{B}, \mathit{true}, \wedge)$ it forms the conjunction of a list of Booleans. Its type is $\mathsf{Free}\;A \to (A, e, \bullet)$, which is equivalent to $\mathsf{Free}\,(\mathsf{U}\,(A, e, \bullet)) \to (A, e, \bullet)$. And, indeed, reduce is natural in the monoid $(A, e, \bullet)$, that is, $\mathit{reduce} : \mathsf{Free} \circ \mathsf{U} \mathrel{\dot{\to}} \mathsf{Id}$. Given $h : (A, 0, +) \to (B, 1, \times)$, the naturality condition is

$$h \cdot \mathit{reduce}\,(A, 0, +) = \mathit{reduce}\,(B, 1, \times) \cdot \mathsf{Free}\,(\mathsf{U}\,h).$$

This says that reducing a list using monoid A and then converting it to monoid B using h is the same thing as converting all the elements of the list to B using h, and then reducing the list using monoid B.

An application of *reduce* worth singling out is *join*, which flattens a list of lists of elements (recall that $\mathsf{List} = \mathsf{U} \circ \mathsf{Free}$):

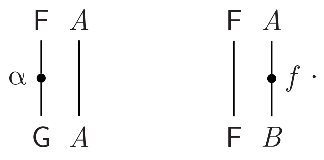
$$\mathit{join}\,A := \mathsf{U}\,(\mathit{reduce}\,(\mathsf{Free}\,A)) : \mathsf{U}\,(\mathsf{Free}\,(\mathsf{U}\,(\mathsf{Free}\,A))) \to \mathsf{U}\,(\mathsf{Free}\,A).$$

The function is natural in A, that is, $\mathit{join} : \mathsf{List} \circ \mathsf{List} \mathrel{\dot{\to}} \mathsf{List}$. □

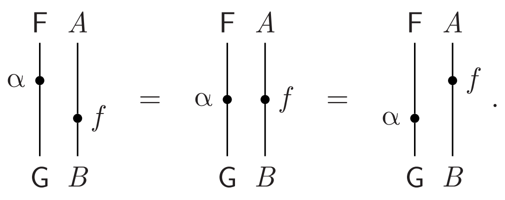
Example 1.29 (Category of Actions)**.** Continuing Example 1.24, a natural transformation between functors of type $\mathcal{M} \to \mathbf{Set}$ is morally the same as an equivariant map; see Example 1.5. The coherence condition (1.3) corresponds to the naturality square (1.15). Overall, the category M-**Act** of actions is isomorphic to the functor category $\mathbf{Set}^{\mathcal{M}}$. □

Example 1.30 (Nonnatural Transformation)**.** We have observed in the introduction that forming a singleton list is a natural transformation of type $\mathsf{Id} \mathrel{\dot{\to}} \mathsf{List}$. There is, however, no natural transformation of type $\mathsf{List} \mathrel{\dot{\to}} \mathsf{Id}$, as there is no *natural* way to define the image of the empty list. In a sense, we have to invent an element of the target type. This cannot be done uniformly – for the target type 0 this is even impossible. □

1.5.2 Graphical Representation of Natural Transformations. Turning to the graphical representation, we depict the component $\alpha\,A$ of a natural transformation, as on the left in the following diagram:



The diagrams for $\alpha\,A$ and $\mathsf{F}\,f$ exhibit a nice symmetry: both consist of two parallel lines, one of which has a "bead" on it. We have noted above that the graphical notation silently deals with applications of the functor laws. The same holds true of the naturality condition (1.15) if we agree that diagrams that differ only in the relative vertical position of "beads," arrows and natural transformations, are identified:



(1.16)

This convention is a natural one, as the two strings are drawn in parallel, suggesting that the corresponding actions are independent of each other.

1.6 Properties of Functors

Like for arrows, we now consider categorical generalizations of injective, surjective, and bijective functions.

Since a functor is an arrow in **Cat**, it can be a (split) mono or a (split) epi. However, often weaker properties are more useful. Recall that a functor $\mathsf{F} : \mathcal{C} \to \mathcal{D}$ consists of an object and an arrow map. If the former is injective (surjective), then F is said to be *injective* (*surjective*) *on objects*. If the latter is injective (surjective), then F is said to be *injective* (*surjective*) *on arrows*.

Two objects that exhibit exactly the same relationships cannot be distinguished: they are isomorphic. Isomorphism is a better notion of "sameness" for objects, rather than equality. The categorically natural properties of objects, such as being a terminal object, a product, or an exponential, are carried across isomorphisms. Consequently, we obtain more appropriate notions of "injective (surjective) on objects" if we replace equality by isomorphism.

1.6.1 Essentially Injective and Surjective Functors. A functor $\mathsf{F} : \mathcal{C} \to \mathcal{D}$ is *essentially injective* (on objects) if

$$\mathsf{F}\,X_1 \cong \mathsf{F}\,X_2 \quad\Longrightarrow\quad X_1 \cong X_2, \tag{1.17a}$$

for all objects $X_1, X_2 : \mathcal{C}$.

A functor $\mathsf{F} : \mathcal{C} \to \mathcal{D}$ is called *essentially surjective* (on objects) if

$$\forall B : \mathcal{D}\;.\;\exists A : \mathcal{C}\;.\;B \cong \mathsf{F}\,A. \tag{1.17b}$$

A functor is *essentially bijective* if it is both essentially injective and essentially surjective.

1.6.2 Faithful and Full Functors. While the object map of a functor is a single map, the arrow map is really a family of maps:

$$\mathsf{F}_{A,B} : (A \to B : \mathcal{C}) \to (\mathsf{F}\,A \to \mathsf{F}\,B : \mathcal{D}),$$

with one member for each pair of objects. In fact, these maps are natural in both A and B, meaning that the following equation holds:

$$\mathsf{F}\,k \cdot \mathsf{F}_{A,B}\,f \cdot \mathsf{F}\,h = \mathsf{F}_{A',B'}\,(k \cdot f \cdot h),$$

which is an immediate consequence of the second functor law.

The functor F is *faithful* (*full*) if each of these maps is injective (surjective):

$$\mathsf{F}\text{ faithful} \iff \mathsf{F}_{A,B}\text{ injective for all }A\text{ and }B, \tag{1.18a}$$

$$\mathsf{F}\text{ full} \iff \mathsf{F}_{A,B}\text{ surjective for all }A\text{ and }B. \tag{1.18b}$$

A functor that is injective on arrows is also faithful, but not necessarily the other way round; see Exercise 1.31. For a subcategory $\mathcal{S}$ of $\mathcal{C}$, the inclusion functor $\mathcal{S} \to \mathcal{C}$ is both faithful and injective on objects. It is furthermore full if and only if $\mathcal{S}$ is a full subcategory.

For a functor that is *fully faithful* (full and faithful), the maps $\mathsf{F}_{A,B}$ form a natural bijection:

$$A \to B \quad \cong \quad \mathsf{F}\,A \to \mathsf{F}\,B. \tag{1.19}$$

A functor preserves isomorphisms; a fully faithful functor also reflects isomorphisms:

$$A \cong B \quad \Longleftrightarrow \quad \mathsf{F}\,A \cong \mathsf{F}\,B. \tag{1.20}$$

In other words, a fully faithful functor is essentially injective on objects.

1.7 Equational Reasoning

Category theory is essentially an algebraic theory: propositions take the form of equations, whose proofs are conducted using equational reasoning. Equational proofs are attractive for several reasons. First and foremost they are simple in the sense that they do not involve a lot of machinery – the basic step is to replace equals by equals. To establish $f = g$ one seeks intermediate terms h_0, ..., h_n such that $f = h_0 = \cdots = h_n = g$.

1.7.1 Symbolic Calculational Proofs. The following proof format, attributed to Wim Feijen (Gasteren, van, 1988, p. 107), is used throughout the monograph:

$$\begin{array}{ll} & \mathit{term}_1 \\ = & \{\ \text{hint 1}\ \} \\ & \mathit{term}_2 \\ = & \{\ \text{hint 2}\ \} \\ & \mathit{term}_3. \end{array}$$

Each step of the calculation is justified by a hint, enclosed in curly braces. The hints should enable the reader to easily verify that the calculation constitutes a valid proof.

It is instructive to work through a concrete example. (We will revisit the example in Section 2.4.1 once our graphical calculus is in place. Section 3.5 generalizes the example to a more abstract setting.) A common list-processing function is $\mathit{filter}\ p : \mathsf{List}\ A \to \mathsf{List}\ A$, which takes a list as an argument and returns the list of all those elements, in order, that satisfy the predicate $p : A \to \mathbb{B}$. It is defined as

$$\mathit{filter}\, p := \mathit{join}\, A \cdot \mathsf{List}\,(\mathit{guard}\, p), \tag{1.21}$$

where $\mathit{guard}\, p : A \to \mathsf{List}\, A$ takes an element to a singleton list, if the element satisfies p, or to the empty list otherwise.

Our goal is to show that *filter* satisfies the following property:

$$\mathit{filter}\, p \cdot \mathit{join}\, A = \mathit{join}\, A \cdot \mathsf{List}\,(\mathit{filter}\, p), \tag{1.22}$$

for all objects A and for all arrows $p : A \to \mathbb{B}$. The equation can be paraphrased as follows: given a list of lists it does not matter whether we first flatten the lists and then filter the result, or we first filter the element lists individually and then flatten the filtered lists.

For the proof of (1.22) we need a fundamental property of flattening:

$$\mathit{join}\, A \cdot \mathsf{List}\,(\mathit{join}\, A) = \mathit{join}\, A \cdot \mathit{join}\,(\mathsf{List}\, A), \tag{1.23}$$

which states that the two ways of flattening a list of lists of lists of elements (a cubed list) are equivalent. We reason,

$$\begin{aligned}
& \mathit{filter}\, p \cdot \mathit{join}\, A \\
= {} & \quad \{ \text{ definition of } \mathit{filter} \text{ (1.21) } \} \\
& \mathit{join}\, A \cdot \mathsf{List}\,(\mathit{guard}\, p) \cdot \mathit{join}\, A \\
= {} & \quad \{ \ \mathit{join} \text{ is natural (1.15) } \} \\
& \mathit{join}\, A \cdot \mathit{join}\,(\mathsf{List}\, A) \cdot \mathsf{List}\,(\mathsf{List}\,(\mathit{guard}\, p)) \\
= {} & \quad \{ \text{ property of } \mathit{join} \text{ (1.23) } \} \\
& \mathit{join}\, A \cdot \mathsf{List}\,(\mathit{join}\, A) \cdot \mathsf{List}\,(\mathsf{List}\,(\mathit{guard}\, p)) \\
= {} & \quad \{ \ \mathsf{List} \text{ is a functor (1.11b) } \} \\
& \mathit{join}\, A \cdot \mathsf{List}\,(\mathit{join}\, A \cdot \mathsf{List}\,(\mathit{guard}\, p)) \\
= {} & \quad \{ \text{ definition of } \mathit{filter} \text{ (1.21) } \} \\
& \mathit{join}\, A \cdot \mathsf{List}\,(\mathit{filter}\, p).
\end{aligned}$$

Observe that the proof makes implicit use of Leibniz's context rules; see Section 1.2.

When writing or reading an equational proof, it is customary or even advisable to start at both ends, applying some obvious rewrites such as unfolding definitions. In the preceding example, we note that the loose ends can be connected by applying Property (1.23).

The previous calculation exemplifies an equational proof. The proof format works equally well for arbitrary *transitive* relations, for example $\leqslant$, $<$, or $\Longrightarrow$. The following calculation demonstrates that f is mono if $g \cdot f$ is mono:

$$\begin{aligned}
& f \cdot x_1 = f \cdot x_2 \\
\Longrightarrow\quad & \{\ \text{Leibniz (1.5c)}\ \} \\
& g \cdot f \cdot x_1 = g \cdot f \cdot x_2 \\
\Longrightarrow\quad & \{\ \text{assumption: } g \cdot f \text{ mono (1.5a)}\ \} \\
& x_1 = x_2 .
\end{aligned}$$

1.7.2 Commutative Diagrams. Category theory has a strong visual flavor. Diagrams can be used to visualize not only arrows, but also properties of arrows, and to conduct proofs. We shall see that diagrammatic proof is an attractive alternative to symbolic proof.

Recall that a composite arrow can be visualized by a one-dimensional drawing of a path. To visualize an equality between two composite arrows, we can connect the corresponding paths at both ends, obtaining a two-dimensional drawing. As an example, the following equation on the left can be represented by the diagram on the right:

$$\begin{aligned}
& g \cdot f \\
=\quad & \{\ \text{hint}\ \} \\
& k \cdot h
\end{aligned}
\qquad\qquad
\begin{tikzcd}
 & X \arrow[dl, "g"'] & \\
B & \text{hint} & A \arrow[ul, "f"'] \arrow[dl, "h"] \\
 & Y \arrow[ul, "k"] &
\end{tikzcd} .$$

As noted before, the advantage of diagrammatic over symbolic notation is that it adds vital type information.

A diagram where all paths from the same source to the same target lead to the same result by composition is called a *commutative diagram*. A commutative diagram can represent an equation (see the definition of products and coproducts), but it can also serve as a proof. To illustrate, here is a proof of (1.22) framed as a commutative diagram. (For clarity, the unfolding of definitions is omitted and List is abbreviated to L.)

$$\begin{tikzcd}
 & \mathsf{L}(\mathsf{L}A) \arrow[dl, "join\,A"'] & & \mathsf{L}A \arrow[ll, "\mathsf{L}(guard\,p)"'] \arrow[dl, phantom, "(1.15)"] & \\
\mathsf{L}A & (1.23) & \mathsf{L}(\mathsf{L}(\mathsf{L}A)) \arrow[ul, "join(\mathsf{L}A)"'] \arrow[dl, "\mathsf{L}(join\,A)"] & & \mathsf{L}(\mathsf{L}A) \arrow[ul, "join\,A"'] \arrow[ll, "\mathsf{L}(\mathsf{L}(guard\,p))"'] \arrow[dlll, bend left, "\mathsf{L}(join\,A\cdot\mathsf{L}(guard\,p))"] \arrow[dll, phantom, "(1.11b)"] \\
 & \mathsf{L}(\mathsf{L}A) \arrow[ul, "join\,A"] & & &
\end{tikzcd}$$

The outer pentagon represents the conclusion, the to-be-shown equation; the inner squares and triangles constitute the assumptions. One can show that, for the entire diagram to commute, it is sufficient that the inner diagrams commute.

Commutative diagrams work well for objects and arrows, but less so for categories, functors, and natural transformations. This is already evident in the previous example, where complex terms are used as labels, mixing symbolic and diagrammatic notation. We argue that string diagrams, developed in Chapter 2, are a better alternative.

Summary

A category consists of two components, objects and arrows. Functors are structure-preserving maps between categories, and natural transformations can be seen as mappings between functors. Categories, functors, and natural transformations form a so-called 2-category.

Categories generalize both monoids and preorders:

in **Cat**	in **Mon**	in **Pre**
category	monoid	preorder
functor	monoid homomorphism	monotone map
natural transformation	pointwise conjugation	pointwise preorder

There are many ways of constructing new categories from old: subcategories, opposite categories, product categories, and functor categories. Opposite categories are at the heart of the duality principle: categorical concepts come in pairs; duality cuts down the work by half. Product categories allow us to capture n-ary functors, and functor categories higher-order functors – functors that take functors as arguments or yield functors as results.

Further Reading

The basic concepts of category were introduced by Eilenberg and MacLane (1945), and this article is still very relevant.

The standard modern reference for basic category theory is the excellent Mac Lane (1998), although it does require a certain amount of mathematical experience to understand the examples. Stronger expository accounts are Awodey (2010), Leinster (2014), and the more computer science oriented Crole (1993). The recent Spivak (2014) is written specifically with practitioners outside mathematics in mind.

Modern encyclopedic accounts of large parts of category theory include Borceux (1994a,b,c) and Johnstone (2002a,b).

We use order theory and lattices in many of our examples. An enjoyable introduction to the fundamentals is Davey and Priestley (2002). Backhouse et al. (1998) provided an explicit development of category theory as a generalization of order theoretic notions.

Exercises

1.1 ○ Summarize the contents of this chapter in your own words.

1.2 ● Precisely define the categories mentioned in Section 1.1. What are the objects, what are the arrows? How are the identity arrows and composition of arrows defined?

1.3 ⦿ Define a category whose objects are natural numbers and whose arrows of type $n \to m$ are real-valued matrices of dimension $m \times n$.

1.4 ⦿ The category **Rel** features relations as arrows. Relations can also be part of the object data. Consider triples (A_1, R, A_2), where A_1 and A_2 are sets and $R \subseteq A_1 \times A_2$ is a binary relation. These triples form the objects of a category; an arrow between objects (A_1, R, A_2) and (B_1, S, B_2) is then a pair of functions (f_1, f_2) with $f_1 : A_1 \to B_1$ and $f_2 : A_2 \to B_2$ such that

$$\forall a_1 \in A_1 \,.\, \forall a_2 \in A_2 \,.\, (a_1, a_2) \in R \implies (f_1\, a_1, f_2\, a_2) \in S.$$

In words, the functions take related arguments to related results. Fill in the details by finding a suitable notion of composition, and identity arrows. Show that composition is both associative and unital. This category underlies Wadler's "Theorems for Free!" (Wadler, 1989).

1.5 ○ Are **Mon** and **Pre** subcategories of **Cat**? If yes, are they full?

1.6 ○ (a) Let $\mathcal{C}$ be a monoid category. What is the opposite category $\mathcal{C}^{op}$?
(b) Let $\mathcal{C}$ be a preorder category. What is the opposite category $\mathcal{C}^{op}$?

1.7 ⊙ (a) Let $\mathcal{C}$ and $\mathcal{D}$ be two monoid categories. What is the product category $\mathcal{C} \times \mathcal{D}$? (b) Let $\mathcal{C}$ and $\mathcal{D}$ be two preorder categories. What is the product category $\mathcal{C} \times \mathcal{D}$?

1.8 ⊙ Prove that g is split epi if and only if the partial application $g \cdot -$ is surjective (1.6c). Also show the dual statement (1.6d).

1.9 ○ Show that, if f has both a pre- and post-inverse, they must coincide.

1.10 ⊙ (a) Show that the isos in **Mon** are exactly the bijective monoid homomorphisms. (b) Show that the isos in **Pre** are *not* the same as bijective, monotone functions.

1.11 ● Prove the relations summarized in Figure 1.1 and show that the inclusions are proper: prove that a split mono is a mono and exhibit a mono that is not a split mono, and so on.

1.12 ◉ Explain why the notions of mono, epi, split mono, and split epi give rise to only seven different concepts (which ones?), even though there are $2^4 = 16$ different combinations of four properties.

1.13 ○ Find a category with multiple distinct initial and final objects.

1.14 ⊙ Show that the product of B_1 and B_2 is unique up to a *unique* isomorphism that makes the diagram

$$\begin{tikzcd}
 & B_1 & \\
B_1 \times B_2 \arrow[ur, "outl"] \arrow[dr, "outr"'] \arrow[rr, dotted, shift left] \arrow[rr, phantom, "\cong"] & & B_1 \times' B_2 \arrow[ul, "outl'"'] \arrow[dl, "outr'"] \arrow[ll, dotted, shift left] \\
 & B_2 &
\end{tikzcd}$$

commute. (It is *not* the case that there is a unique isomorphism *per se*. For example, there are two isomorphisms between $B \times B$ and $B \times B$: the identity $id_{B \times B} = outl \vartriangle outr$ and $outr \vartriangle outl$.)

1.15 ⊙ Define the coproduct of two categories, dualizing the product of categories; see Definition (1.8). *Hint:* tag the objects and arrows; see Example 1.13. You must find a sensible way to define identity and composition.

1.16 ⊙ *Hint:* you need to solve Exercise 1.15 first. (a) Let $\mathcal{C}$ and $\mathcal{D}$ be two monoid categories. What is the coproduct category $\mathcal{C} + \mathcal{D}$? (b) Let $\mathcal{C}$ and $\mathcal{D}$ be two preorder categories. What is the coproduct category $\mathcal{C} + \mathcal{D}$?

1.17 ● Linear maps between two vector spaces form again a vector space. Does this imply that **Vect** $(\mathbb{K})$ has exponentials? *Hint:* you may want to peek at Exercise 4.7 first.

1.18 ◉ Does **Mon** have initial and final objects? What about coproducts and products (see also Exercises 1.7(a) and 1.16(a))? And exponentials (see also Exercise 1.25(a))?

1.19 ⦿ Does **Pre** have initial and final objects? What about coproducts and products (see also Exercises 1.7(b) and 1.16(b))? And exponentials (see also Exercise 1.25(b))?

1.20 ⊙ Does **Rel** have initial and final objects? What about coproducts and products? And exponentials?

1.21 ⊙ This exercise aims to fill in some of the details of Example 1.21. Let A be a set, $\bullet$ a binary function on A, and e a fixed element in A. Define:

$$x \leqslant y \iff x \bullet y = y.$$

Show that:

- $\leqslant$ is reflexive if $\bullet$ is idempotent.
- $\leqslant$ is antisymmetric if $\bullet$ is commutative.
- $\leqslant$ is transitive if $\bullet$ is associative.
- If e is the unit element, then it is the least element of the order $\leqslant$.
- If $h : (A, \bullet, e) \to (B, \bullet, e)$ is a homomorphism of idempotent monoids, show that it is monotone with respect to $\leqslant$.

1.22 ⊙ Show diagrammatically that split monos and split epis are preserved by functor application. Conclude that functors preserve isomorphisms:

$$A \cong B \implies \mathsf{F}\, A \cong \mathsf{F}\, B.$$

Are all monos and epis preserved by functors?

1.23 ⦿ If we fix one argument of a bifunctor, we obtain a functor. The converse is not true: functoriality in each argument separately does not imply functoriality in both. Rather, we have the following: the map $- \otimes = : \mathcal{C} \times \mathcal{D} \to \mathcal{E}$ is a bifunctor if and only if the partial application $- \otimes B : \mathcal{C} \to \mathcal{E}$ is a functor for all $B : \mathcal{D}$, the partial application $A \otimes - : \mathcal{D} \to \mathcal{E}$ is a functor for all $A : \mathcal{C}$, and if furthermore the two collections of unary functors satisfy the *exchange condition*,

$$(f \otimes B'') \cdot (A' \otimes g) = (A'' \otimes g) \cdot (f \otimes B'), \tag{1.24}$$

for all $f : A' \to A'' : \mathcal{C}$ and $g : B' \to B'' : \mathcal{D}$. Given f and g there are two ways of turning $A' \otimes B'$ things into $A'' \otimes B''$ things. The exchange condition (1.24) demands that they are equal:

$$\begin{tikzcd}
A' \otimes B'' & A' \otimes B' \arrow[l, "A'\otimes g"'] \arrow[d, "f\otimes B'"] \arrow[dl, "f\otimes g"'] \\
A'' \otimes B'' & A'' \otimes B' \arrow[l, "A''\otimes g"]
\arrow[from=1-1, to=2-1, "f\otimes B''"']
\end{tikzcd} .$$

The arrow part of the bifunctor, the diagonal, is then given by either side of the equation. Prove the so-called exchange lemma.

1.24 ⊙ Fill in the details of Example 1.19. In particular, show that the arrow map of the bifunctor $- \times =$ satisfies the typing requirements and the functor laws. Dualize to coproducts. Show that $(-)^X$ is functorial.

1.25 ⊙ (a) Let $\mathcal{C}$ and $\mathcal{D}$ be two monoid categories. What is the functor category $\mathcal{D}^{\mathcal{C}}$? (b) Let $\mathcal{C}$ and $\mathcal{D}$ be two preorder categories. What is the functor category $\mathcal{D}^{\mathcal{C}}$?

1.26 ○ Continuing Exercise 1.24, show that *outl* and *outr* are natural transformations. Dualize to coproducts. Show that $apply : B^X \times X \to B$ is natural in B.

1.27 ⊙ How many natural transformations are there of type $\mathsf{Id} \stackrel{\cdot}{\to} \mathsf{Id}$, where $\mathsf{Id} : \mathbf{Set} \to \mathbf{Set}$? And of type $\mathsf{Id} \stackrel{\cdot}{\to} \mathsf{P}$, where $\mathsf{P}\,A = A \times A$? And if we flip source and target: $\mathsf{P} \stackrel{\cdot}{\to} \mathsf{Id}$?

1.28 ⊙ Continuing Exercise 1.23 show that functor application and functor composition are bifunctors:

$$\mathsf{Apply} : \mathcal{D}^{\mathcal{C}} \times \mathcal{C} \to \mathcal{D} \qquad \text{and} \qquad - \circ = : \mathcal{E}^{\mathcal{D}} \times \mathcal{D}^{\mathcal{C}} \to \mathcal{E}^{\mathcal{C}} .$$

1.29 ⊙ Does **Cat** have initial and final objects? What about coproducts and products? And exponentials?

1.30 ◉ Show that the following are equivalent for a category $\mathcal{C}$:

- For every category $\mathcal{B}$ and pair of functors $\mathsf{F}, \mathsf{G} : \mathcal{B} \to \mathcal{C}$, every transformation between F and G is natural.
- $\mathcal{C}$ is a preorder category.

1.31 ◉ Give an example of a functor that is fully faithful, but not injective on arrows. *Hint:* solve Exercise 1.15 first.

1.32 ⊙ Show the following implications:

$$\begin{aligned}
\mathsf{F} \circ \mathsf{G} \text{ is essentially surjective} &\implies \mathsf{F} \text{ is essentially surjective,} \\
\mathsf{F} \circ \mathsf{G} \text{ is faithful} &\implies \mathsf{G} \text{ is faithful.}
\end{aligned}$$

1.33 ⊙ Show that $h : \mathsf{List}\,A \to \mathsf{List}\,B$ where

$$h \cdot \mathit{join}\,A = \mathit{join}\,B \cdot \mathsf{List}\,h$$

is a monoid homomorphism over the free monoid, assuming suitable properties of *join*. Conclude that *filter* is a homomorphism.