Category Theory
Lecture Notes
for ESSLLI

Michael Barr
Department of Mathematics and Statistics
McGill University

Charles Wells
Department of Mathematics
Case Western Reserve University

Contents

Preface

About these notes

These notes form a short summary of some major topics in category theory. They are a condensation (with some rearrangement) of part of [Barr and Wells, 1999]. The chapter and section numbers in the notes are not the same as those in the book.

About categories

Categories originally arose in mathematics out of the need of a formalism to describe the passage from one type of mathematical structure to another. A category in this way represents a kind of mathematics, and may be described as **category as mathematical workspace**.

A category is also a mathematical structure. As such, it is a common generalization of both ordered sets and monoids (the latter are a simple type of algebraic structure that include transition systems as examples), and questions motivated by those topics often have interesting answers for categories. This is **category as mathematical structure**.

Finally, a category can be seen as a structure that formalizes a mathematician's description of a type of structure. This is the role of **category as theory**. Formal descriptions in mathematical logic are traditionally given as formal languages with rules for forming terms, axioms and equations. Algebraists long ago invented a formalism based on tuples, the method of signatures and equations, to describe algebraic structures. Category theory provides another approach: the category is a theory and functors with that category as domain are models of the theory.

Other categorical literature

All of the topics in these notes are covered in more depth, with applications to computing science, in our text [Barr and Wells, 1999]. Most of the topics are also developed further, but without applications to computing science, in [Barr and Wells, 1985], [Mac Lane, 1971], [McLarty, 1992], [Freyd and Scedrov, 1990] and [Borceux, 1994] Other texts specifically concerning applications to computing science include [Asperti and Longo, 1991], [Crole, 1994], [Gunter, 1992], [Manes and Arbib, 1986], [Pierce, 1991], [Rydeheard and Burstall, 1988] and [Walters, 1991]. Various aspects of the close relationship between logic and categories (in their role as theories) are treated in [Makkai and Reyes, 1977], [Lambek and Scott, 1986], [Bell, 1988] and [Adámek and Rosičky, 1994]. Recent collections of papers in computer science which have many applications of category theory are [Pitt *et al.*, 1986], [Pitt, Poigné and Rydeheard, 1987], [Ehrig *et al.*, 1988], [Main *et al.*, 1988], [Gray and Scedrov, 1989], [Pitt *et al.*, 1989], [Pitt *et al.*, 1991], [Fourman, Johnstone and Pitts, 1992], [Seely, 1992], [Pitt, Rydeheard and Johnstone, 1995] and [Moggi and Rosolini, 1997]. The reader may find useful the discussions of the uses of category theory in computing science in [Goguen, 1991], [Fokkinga, 1992] and in the tutorials in [Pitt *et al.*, 1986].

Michael Barr
Department of Mathematics
and Statistics
McGill University
805 Sherbrooke St. W.
Montréal, Québec
Canada H3P 1S4
barr@barrs.org

Charles Wells
Department of Mathematics
Case Western Reserve University
10900 Euclid Ave
Cleveland, OH 44106-7058
USA
charles@freude.com

http://www.math.mcgill.ca/~barr
http://www.cwru.edu/artsci/math/wells/home.html

1. Preliminaries

1.1 Graphs

The type of graph that we discuss in this section is a specific version of directed graph, one that is well adapted to category theory, namely what is often called a directed multigraph with loops. A graph is an essential ingredient in the definition of commutative diagram, which is the categorist's way of expressing equations. The concept of graph is also a precursor to the concept of category itself: a category is, roughly speaking, a graph in which paths can be composed.

1.1.1 Definition and notation Formally, to specify a **graph**, you must specify its **nodes** (or **objects**) and its **arrows**. Each arrow must have a specific **source** (or **domain**) node and **target** (or **codomain**) node. The notation '$f : a \longrightarrow b$' means that f is an arrow and a and b are its source and target, respectively. If the graph is small enough, it may be drawn with its nodes indicated by dots or labels and each arrow by an actual arrow drawn from its source to its target.

There may be one or more arrows – or none at all – with given nodes as source and target. Moreover, the source and target of a given arrow need not be distinct. An arrow with the same source and target node will be called an **endoarrow** or **endomorphism** of that node.

We will systematically denote the collection of nodes of a graph $\mathscr{G}$ by G_0 and the collection of arrows by G_1, and similarly with other letters ($\mathscr{H}$ has nodes H_0, $\mathscr{C}$ has nodes C_0, and so on). The nodes form the zero-dimensional part of the graph and the arrows the one-dimensional part.

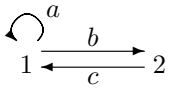
1.1.2 Example Let $G_0 = \{1, 2\}$, $G_1 = \{a, b, c\}$,

$$\text{source}(a) = \text{target}(a) = \text{source}(b) = \text{target}(c) = 1$$

and

$$\text{target}(b) = \text{source}(c) = 2$$

Then we can represent $\mathscr{G}$ as



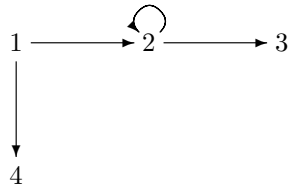
(1.1)

1.1.3 Example The **graph of sets and functions** has all sets as nodes and all functions between sets as arrows. The source of a function is its domain, and its target is its codomain. This is a **large** graph: because of Russell's paradox, its nodes and its arrows do not form sets.

1.1.4 Example It is often convenient to picture a relation on a set as a graph. For example, let $A = \{1, 2, 3\}$, $B = \{2, 3, 4\}$ and

$$\alpha = \{(1, 2), (2, 2), (2, 3), (1, 4)\}$$

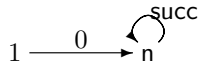
Then α can be pictured as



(1.2)

Of course, graphs that arise this way never have more than one arrow with the same source and target. Such graphs are called **simple graphs**.

1.1.5 Example A data structure can sometimes be represented by a graph. This graph represents the set **N** of natural numbers in terms of zero and the successor function (adding 1):



(1.3)

The name '1' for the left node is the conventional notation to require that the node denote a **singleton set**, that is, a set with exactly one element. One can give a formal mathematical meaning to the idea that this graph generates the natural numbers; see [Barr and Wells, 1999], Section 4.7.6.

This informal idea of a graph representing a data type is the basis of the formal theory of sketches, developed in [Barr and Wells, 1999]. In particular, Section 4.7.6 of that text gives a formal mathematical meaning to the idea that the graph just given generates the natural numbers.

1.2 Homomorphisms of graphs

A homomorphism of graphs should preserve the abstract shape of the graph. A reasonable translation of this vague requirement into a precise mathematical definition is as follows.

1.2.1 Definition A **homomorphism** ϕ from a graph $\mathscr{G}$ to a graph $\mathscr{H}$, denoted $\phi : \mathscr{G} \longrightarrow \mathscr{H}$, is a pair of functions $\phi_0 : G_0 \longrightarrow H_0$ and $\phi_1 : G_1 \longrightarrow H_1$ with the property that if $u : m \longrightarrow n$ is an arrow of $\mathscr{G}$, then $\phi_1(u) : \phi_0(m) \longrightarrow \phi_0(n)$ in $\mathscr{H}$.

It is instructive to restate this definition using the source and target mappings: let $\text{source}_{\mathscr{G}} : G_1 \longrightarrow G_0$ be the source map that takes an arrow (element of G_1) to its source, and similarly define $\text{target}_{\mathscr{G}}$, $\text{source}_{\mathscr{H}}$ and $\text{target}_{\mathscr{H}}$. Then the pair of maps $\phi_0 : G_0 \longrightarrow H_0$ and $\phi_1 : G_1 \longrightarrow H_1$ is a graph homomorphism if and only if

$$\text{source}_{\mathscr{H}} \circ \phi_1 = \phi_0 \circ \text{source}_{\mathscr{G}}$$

and

$$\text{target}_{\mathscr{H}} \circ \phi_1 = \phi_0 \circ \text{target}_{\mathscr{G}}$$

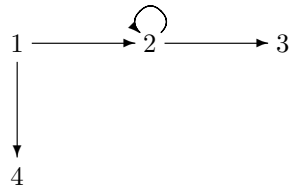
1.2.2 Notation of the form $a : B \longrightarrow C$ is overloaded in several ways. It can denote a set-theoretic function, a graph homomorphism or an arrow in a graph. In fact, all three are instances of the third since there is a large graph whose nodes are sets and arrows are functions (see 1.1.3) and another whose nodes are (small) graphs and arrows are graph homomorphisms.

Another form of overloading is that if $\phi : \mathscr{G} \longrightarrow \mathscr{H}$ is a graph homomorphism, ϕ actually stands for a pair of functions we here call $\phi_0 : G_0 \longrightarrow H_0$ and $\phi_1 : G_1 \longrightarrow H_1$. In fact, it is customary to omit the subscripts and use ϕ for all three (the graph homomorphism as well as its components ϕ_0 and ϕ_1).

This does not lead to ambiguity in practice; in reading about graphs you are nearly always aware of whether the author is talking about nodes or arrows. We will keep the subscripts in this section and drop them thereafter.

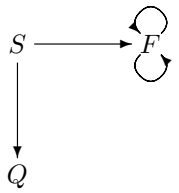
1.2.3 Example If $\mathscr{G}$ is any graph, the **identity homomorphism** $\text{id}_{\mathscr{G}} : \mathscr{G} \longrightarrow \mathscr{G}$ is defined by $(\text{id}_{\mathscr{G}})_0 = \text{id}_{G_0}$ (the identity function on the set of nodes of $\mathscr{G}$) and $(\text{id}_{\mathscr{G}})_1 = \text{id}_{G_1}$.

1.2.4 Example If $\mathscr{G}$ is the graph



(1.4)

and $\mathscr{H}$ is this graph,



(1.5)

then there is a homomorphism $\phi : \mathscr{G} \to \mathscr{H}$ for which $\phi_0(1) = S$, $\phi_0(2) = \phi_0(3) = F$ and $\phi_0(4) = Q$, and ϕ_1 takes the loop on 2 and the arrow from 2 to 3 both to the upper loop on F; what ϕ_1 does to the other two arrows is forced by the definition of homomorphism. Because there are two loops on F there are actually four possibilities for ϕ_1 on arrows (while keeping ϕ_0 fixed).

1.2.5 Example If $\mathscr{H}$ is any graph with a node n and a loop $u : n \to n$, then there is a homomorphism from *any* graph $\mathscr{G}$ to $\mathscr{H}$ that takes every node of $\mathscr{G}$ to n and every arrow to u. This construction gives two other homomorphisms from $\mathscr{G}$ to $\mathscr{H}$ in Example 1.2.4 besides the four mentioned there. (There are still others.)

1.2.6 Notation In an expression like '$\phi_1(\text{source})(f)$', ϕ_1 is a function whose value at 'source' is a function that applies to an arrow f. As this illustrates, the application operation associates to the left.

2. Categories

A category is a graph with a rule for composing arrows head to tail to give another arrow. This rule is subject to certain conditions, which we will give precisely in Section 2.1. The connection between functional programming languages and categories is described in Section 2.2. Some special types of categories are given in Section 2.3. Sections 2.4 and 2.5 are devoted to a class of examples of the kind that originally motivated category theory. The reader may wish to read through these examples rapidly rather than trying to understand every detail. Constructions that can be made with categories are described in Section 2.6.

2.1 Basic definitions

Before we define categories, we need a preliminary definition.

2.1.1 Definition Let $k > 0$. In a graph $\mathscr{G}$, a **path** from a node x to a node y of length k is a sequence $(f_1, f_2, \ldots, f_k)$ of (not necessarily distinct) arrows for which

(i) $\text{source}(f_k) = x$,
(ii) $\text{target}(f_i) = \text{source}(f_{i-1})$ for $i = 2, \ldots, k$, and
(iii) $\text{target}(f_1) = y$.

By convention, for each node x there is a unique path of length 0 from x to x that is denoted (). It is called the **empty path** at x.

Observe that if you draw a path as follows:

$$\cdot \xrightarrow{f_k} \cdot \xrightarrow{f_{k-1}} \cdots \xrightarrow{f_2} \cdot \xrightarrow{f_1} \cdot$$

with the arrows going from left to right, f_k will be on the left and the subscripts will go down from left to right. We do it this way for consistency with composition (compare 2.1.3, C–1).

For any arrow f, (f) is a path of length 1.

2.1.2 Definition The set of paths of length k in a graph $\mathscr{G}$ is denoted G_k.

In particular, G_2, which will be used in the definition of category, is the set of pairs of arrows (g, f) for which the target of f is the source of g. These are called **composable pairs** of arrows.

> We have now assigned two meanings to G_0 and G_1. This will cause no conflict as G_1 refers indifferently either to the collection of arrows of $\mathscr{G}$ or to the collection of paths of length 1, which is essentially the same thing. Similarly, we use G_0 to represent either the collection of nodes of $\mathscr{G}$ or the collection of empty paths, of which there is one for each node. In each case we are using the same name for two collections that are not the same but are in a natural one to one correspondence. Compare the use of '2' to denote either the integer or the real number. As this last remark suggests, one might want to keep the two meanings of G_1 separate for purposes of implementing a graph as a data structure.
>
> The one to one correspondences mentioned in the preceding paragraph were called 'natural'. The word is used informally here, but in fact these correspondences are natural in the technical sense (See Section 5.3).

2.1.3 Categories A **category** is a graph $\mathscr{C}$ together with two functions $c : C_2 \longrightarrow C_1$ and $u : C_0 \longrightarrow C_1$ with properties C–1 through C–4 below. (Recall that C_2 is the set of paths of length 2.) The elements of C_0 are called **objects** and those of C_1 are called **arrows**. The function c is called **composition**, and if (g, f) is a composable pair, $c(g, f)$ is written $g \circ f$ and is called the **composite** of g and f. If A is an object of $\mathscr{C}$, $u(A)$ is denoted id_A, which is called the **identity** of the object A.

C–1 The source of $g \circ f$ is the source of f and the target of $g \circ f$ is the target of g.

C–2 $(h \circ g) \circ f = h \circ (g \circ f)$ whenever either side is defined.

C–3 The source and target of id_A are both A.

C–4 If $f : A \longrightarrow B$, then $f \circ \mathrm{id}_A = \mathrm{id}_B \circ f = f$.

The significance of the fact that the composite c is defined on G_2 is that $g \circ f$ is defined if and only if the source of g is the target of f. This means that composition is a function whose domain is an equationally defined subset of $G_1 \times G_1$: the equation requires that the source of g equal the target of f. It follows from this and C–1 that in C–2, one side of the equation is defined if and only if the other side is defined.

In the category theory literature, id_A is often written just A.

2.1.4 Terminology In much of the categorical literature, 'morphism', 'domain' and 'codomain' are more common than 'arrow', 'source' and 'target'. In these notes we usually use the language we have just introduced of 'arrow', 'source' and 'target'. We will normally denote objects of categories by capital letters but nodes of graphs (except when we think of a category as a graph) by lower case letters. Arrows are always lower case.

In the computing science literature, the composite $g \circ f$ is sometimes written $f; g$, a notation suggested by the perception of a typed functional programming language as a category (see 2.2.1).

We have presented the concept of category as a two-sorted data structure; the sorts are the objects and the arrows. Categories are sometimes presented as one-sorted – arrows only. The objects can be recovered from the fact that C–3 and C–4 together characterize id_A (not hard to prove), so that there is a one to one correspondence between the objects and the identity arrows id_A.

2.1.5 Definition A category is **small** if its objects and arrows constitute sets; otherwise it is **large**.

The category of sets and functions defined in 2.1.11 below is an example of a large category. Although one must in principle be wary in dealing with large classes, it is not in practice a problem; category theorists have rarely, if ever, run into set-theoretic difficulties.

2.1.6 Definition If A and B are two objects of a category $\mathscr{C}$, then the set of all arrows of $\mathscr{C}$ that have source A and target B is denoted $\mathrm{Hom}_{\mathscr{C}}(A, B)$, or just $\mathrm{Hom}(A, B)$ if the category is clear from context.

Thus for each triple A, B, C of objects, composition induces a function

$$\mathrm{Hom}(B, C) \times \mathrm{Hom}(A, B) \longrightarrow \mathrm{Hom}(A, C)$$

A set of the form $\mathrm{Hom}(A, B)$ is called a **hom set**. Other common notations for $\mathrm{Hom}(A, B)$ are $\mathscr{C}(A, B)$ and $\mathscr{C}(AB)$.

2.1.7 The reference to the *set* of all arrows from A to B constitutes an assumption that they do indeed form a set. A category with the property that $\mathrm{Hom}(A, B)$ is a set for all objects A and B is called **locally small**. All categories in these notes are locally small.

2.1.8 Definition For any path $(f_1, f_2, \ldots, f_n)$ in a category $\mathscr{C}$, define $f_1 \circ f_2 \circ \cdots \circ f_n$ recursively by

$$f_1 \circ f_2 \circ \cdots \circ f_n = (f_1 \circ f_2 \circ \cdots \circ f_{n-1}) \circ f_n, \quad n > 2$$

2.1.9 Proposition The general associative law. *For any path*

$$(f_1, f_2, \ldots, f_n)$$

in a category $\mathscr{C}$ and any integer k with $1 < k < n$,

$$(f_1 \circ \cdots \circ f_k) \circ (f_{k+1} \circ \cdots \circ f_n) = f_1 \circ \cdots \circ f_n$$

In other words, you can unambiguously drop the parentheses.

In this proposition, the notation $f_{k+1} \circ \cdots \circ f_n$ when $k = n - 1$ means simply f_n.

This is a standard fact for associative binary operations (see [Jacobson, 1974], Section 1.4) and can be proved in exactly the same way for categories.

2.1.10 Little categories The smallest category has no objects and (of course) no arrows. The next smallest category has one object and one arrow, which must be the identity arrow. This category may be denoted **1**. Other categories that will be occasionally referred to are the categories $\mathbf{1}+\mathbf{1}$ and **2** illustrated below (the loops are identities). In both cases the choice of the composites is forced.

$\mathbf{1}+\mathbf{1}$ **2** (2.1)

2.1.11 Categories of sets The **category of sets** is the category whose objects are sets and whose arrows are functions with composition of functions for c and the identity function from S to S for id_S. The statement that this is a category amounts to the statements that composition of functions is associative and that each identity function $\mathrm{id}_S : S \longrightarrow S$ satisfies $f \circ \mathrm{id}_S = f$ and $\mathrm{id}_S \circ g = g$ for all f with source S and all g with target S. The fact that composition of functions is associative follows by using the definition of functional composition repeatedly:

$$((h \circ g) \circ f)(x) = (h \circ g)(f(x)) = h(g(f(x))) = h((g \circ f)(x)) = (h \circ (g \circ f))(x)$$

The properties of the identity function follow from the definition.

In this text, the category of sets is denoted **Set**. There are other categories whose objects are sets, as follows.

2.1.12 Definition The **category of finite sets**, denoted **Fin**, is the category whose objects are finite sets and whose arrows are all the functions between finite sets.

2.1.13 Definition A **partial function** from a set S to a set T is a function with domain S_0 and codomain T, where S_0 is some subset of S. The **category Pfn of sets and partial functions** has all sets as objects and all partial functions as arrows. If $f : S \longrightarrow T$ and $g : T \longrightarrow V$ are partial functions with f defined on $S_0 \subseteq S$ and g defined on $T_0 \subseteq T$, the composite $g \circ f : S \longrightarrow V$ is the partial function from S to V defined on the subset $\{x \in S_0 \mid f(x) \in T_0\}$ of S by the requirement $(g \circ f)(x) = g(f(x))$.

It is a worthwhile exercise to check that composition so defined is associative.

2.1.14 Definition Let α be a relation from a set S to a set T and β a relation from T to U. The **composite** $\beta \circ \alpha$ is the relation from S to U defined as follows: If $x \in S$ and $z \in U$, $(x, z) \in \beta \circ \alpha$ if and only if there is an element $y \in T$ for which $(x, y) \in \alpha$ and $(y, z) \in \beta$. With this definition of composition, the **category Rel of sets and relations** has sets as objects and relations as arrows. The identity for a set S is the diagonal relation $\Delta_S = \{(x, x) \mid x \in A\}$.

Other examples of categories whose objects are sets are the category of sets and injective functions and the category of sets and surjective functions.

2.2 Functional programming languages as categories

The intense interest in category theory among researchers in computing science in recent years is due in part to the recognition that the constructions in functional programming languages make a functional programming language look very much like a category. The fact that deduction systems are essentially categories has also been useful in computing science.

In this section we describe the similarities between functional programming languages and categories informally, and discuss some of the technical issues involved in making them precise. Deduction systems are discussed in Section 6.5.

2.2.1 Functional programming languages A functional programming language may be described roughly as one that gives the user some primitive types and operations and some constructors from which one can produce more complicated types and operations.

What a pure functional programming language in this sense does *not* have is variables or assignment statements. One writes a program by applying constructors to the types, constants and functions. 'Running' a program consists of applying such an operator to constants of the input type to obtain a value.

This is functional programming in the sense of the 'function-level programming' of Backus [1981a] and [1981b]. (See also [Williams, 1982].) Another widely held point of view is that functional programming means no assignment statements: variables may appear but are not assigned to. Most languages called functional programming languages (for example Haskell and Miranda) are functional in this sense.

We will discuss Backus-style functional programming languages here. The lambda calculus, with variables, is discussed in Chapter 7.

2.2.2 The category corresponding to a functional programming language A functional programming language has:

FPL–1 Primitive data types, given in the language.

FPL–2 Constants of each type.

FPL–3 Operations, which are functions between the types.

FPL–4 Constructors, which can be applied to data types and operations to produce derived data types and operations of the language.

The language consists of the set of all operations and types derivable from the primitive data types and primitive operations. The word 'primitive' means given in the definition of the language rather than constructed by a constructor. Some authors use the word 'constructor' for the primitive operations.

2.2.3 If we make two assumptions about a functional programming language and one innocuous change, we can see directly that a functional programming language L corresponds in a canonical way to a category $C(L)$.

A–1 We must assume that there is a do-nothing operation id_A for each type A (primitive and constructed). When applied, it does nothing to the data.

A–2 We add to the language an additional type called 1, which has the property that from every type A there is a unique operation to 1. We interpret each constant c of type A as an arrow $c : 1 \to A$. This incorporates the constants into the set of operations; they no longer appear as separate data.

A–3 We assume the language has a composition constructor: take an operation f that takes something of type A as input and produces something of type B, and another operation g that has input of type B and output of type C; then doing one after the other is a derived operation (or program) typically denoted $f; g$, which has input of type A and output of type C.

Functional programming languages generally have do-nothing operations and composition constructors, so A–1 and A–3 fit the concept as it appears in the literature. The language resulting from the change in A–2 is operationally equivalent to the original language.

Composition must be associative in the sense that, if either of $(f; g); h$ or $f; (g; h)$ is defined, then so is the other and they are the same operation. We must also require, for $f : A \to B$, that $f; \mathrm{id}_B$ and $\mathrm{id}_A; f$ are defined and are the same operation as f. That is, we impose the **equations** $f; \mathrm{id}_B = f$ and $\mathrm{id}_A; f = f$ on the language. Both these requirements are reasonable in that in any implementation, the two operations required to be the same would surely do the same thing.

2.2.4 Under those conditions, a functional programming language L has a category structure $C(L)$ for which:

FPC–1 The types of L are the objects of $C(L)$.

FPC–2 The operations (primitive and derived) of L are the arrows of $C(L)$.

FPC–3 The source and target of an arrow are the input and output types of the corresponding operation.

FPC–4 Composition is given by the composition constructor, written in the reverse order.

FPC–5 The identity arrows are the do-nothing operations.

The reader may wish to compare the discussion in [Pitt, 1986].

Observe that $C(L)$ is a *model* of the language, not the language itself. For example, in the category $f; \mathrm{id}_B = f$, but in the language f and $f; \mathrm{id}_B$ are different source programs. This is in contrast to the treatment of languages using context free grammars: a context free grammar generates the actual language.

2.2.5 Example As a concrete example, we will suppose we have a simple such language with three data types, `NAT` (natural numbers), `BOOLEAN` (true or false) and `CHAR` (characters). We give a description of its operations in categorical style.

(i) `NAT` should have a constant $0 : 1 \longrightarrow$ `NAT` and an operation `succ` : `NAT` $\longrightarrow$ `NAT`.

(ii) There should be two constants `true`, `false` : $1 \longrightarrow$ `BOOLEAN` and an operation $\neg$ subject to the equations $\neg \circ$ `true` = `false` and $\neg \circ$ `false` = `true`.

(iii) `CHAR` should have one constant $c : 1 \longrightarrow$ `CHAR` for each desired character c.

(iv) There should be two type conversion operations `ord` : `CHAR` $\longrightarrow$ `NAT` and `chr` : `NAT` $\longrightarrow$ `CHAR`. These are subject to the equation `chr` $\circ$ `ord` $= \mathrm{id}_{\texttt{CHAR}}$. (You can think of `chr` as operating modulo the number of characters, so that it is defined on all natural numbers.)

An example program is the arrow 'next' defined to be the composite `chr` $\circ$ `succ` $\circ$ `ord` : `CHAR` $\longrightarrow$ `CHAR`. It calculates the next character in order. This arrow 'next' is an arrow in the category representing the language, and so is any other composite of a sequence of operations.

2.2.6 The objects of the category $C(L)$ of this language are the types `NAT`, `BOOLEAN`, `CHAR` and 1. Observe that typing is a natural part of the syntax in this approach.

The arrows of $C(L)$ consist of all programs, with two programs being identified if they must be the same because of the equations. For example, the arrow

$$\texttt{chr} \circ \texttt{succ} \circ \texttt{ord} : \texttt{CHAR} \longrightarrow \texttt{CHAR}$$

just mentioned and the arrow

$$\texttt{chr} \circ \texttt{succ} \circ \texttt{ord} \circ \texttt{chr} \circ \texttt{ord} : \texttt{CHAR} \longrightarrow \texttt{CHAR}$$

must be the same because of the equation in (iv).

Observe that `NAT` has constants `succ` $\circ$ `succ` $\circ \ldots \circ$ `succ` $\circ$ 0 where `succ` occurs zero or more times.

Composition in the category is composition of programs. Note that for composition to be well defined, if two composites of primitive operations are equal, then their composites with any other program must be equal. For example, we must have

$$\texttt{ord} \circ (\texttt{chr} \circ \texttt{succ} \circ \texttt{ord}) = \texttt{ord} \circ (\texttt{chr} \circ \texttt{succ} \circ \texttt{ord} \circ \texttt{chr} \circ \texttt{ord})$$

as arrows from `CHAR` to `NAT`. This is handled systematically in [Barr and Wells, 1999], Chapter 3.5, using the quotient construction.

This discussion is incomplete, since at this point we have no way to introduce n-ary operations for $n > 1$, nor do we have a way of specifying the flow of control. The first will be remedied in Section 6.3.13. Approaches to the second question are given in ([Barr and Wells, 1999] (sections 5.6 and 14.2), [Cockett, 1989], [Wagner, 1986a]).

2.3 Mathematical structures as categories

Certain common mathematical structures can be perceived as special types of categories.

2.3.1 Preordered and ordered sets If S is a set, a subset $\alpha \subseteq S \times S$ is called a **binary relation** on S. It is often convenient to write $x\alpha y$ as shorthand for $(x, y) \in \alpha$. We say that α is **reflexive** if $x\alpha x$ for all $x \in S$ and **transitive** if $x\alpha y$ and $y\alpha z$ implies $x\alpha z$ for all x, y, $z \in S$.

A set S with a reflexive, transitive relation α on it is a structure (S, α) called a **preordered set**. This structure determines a category $C(S, \alpha)$ defined as follows.

CO–1 The objects of $C(S, \alpha)$ are the elements of S.

CO–2 If $x, y \in S$ and $x\alpha y$, then $C(S, \alpha)$ has exactly one arrow from x to y, denoted (y, x). (The reader might have expected (x, y) here. This choice of notation fits better with the right-to-left composition that we use. Note that the domain of (y, x) is x and the codomain is y.)

CO–3 If x is not related by α to y there is no arrow from x to y.

The identity arrows of $C(S, \alpha)$ are those of the form (x, x); they belong to α because it is reflexive. The transitive property of α is needed to ensure the existence of the composite described in 2.1.3, so that $(z, y) \circ (y, x) = (z, x)$.

2.3.2 Example The category $C(S, \alpha)$ for $S = \{C, D\}$ and

$$\alpha = \{\langle C, C\rangle, \langle C, D\rangle, \langle D, D\rangle\}$$

is the category **2** exhibited in (2.1), page 6.

2.3.3 Ordered sets A preordered set (S, α) for which α is antisymmetric (that is $x\alpha y$ and $y\alpha x$ imply $x = y$) is called an **ordered set** or **poset** (for 'partially ordered set'). Two examples of posets are $(\mathbf{R}, \leq)$, the real numbers with the usual ordering, and for any set S, the poset $(\mathscr{P}(S), \subseteq)$, the set of subsets of S with inclusion as ordering.

It is often quite useful and suggestive to think of a category as a generalized ordered set, and we will refer to this perception to illuminate constructions we make later.

2.3.4 Semigroups A **semigroup** is a set S together with an associative binary operation $m : S \times S \to S$. The set S is called the **underlying set** of the semigroup.

Normally for s and t in S, $m(s, t)$ is written 'st' and called 'multiplication', but note that it does not have to satisfy the commutative law; that is, we may have $st \neq ts$. A **commutative semigroup** is a semigroup whose multiplication is commutative.

It is standard practice to talk about 'the semigroup S', naming the semigroup by naming its underlying set. This will be done for other mathematical structures such as posets as well. Mathematicians call this practice 'abuse of notation'. It is occasionally necessary to be more precise; that happens in this text in Section 9.1.

2.3.5 Powers We set $s^1 = s$ and, for any positive integer k, $s^k = ss^{k-1}$. Such powers of an element obey the laws $s^k s^n = s^{k+n}$ and $(s^k)^n = s^{kn}$ (for *positive* k and n). On the other hand, the law $(st)^k = s^k t^k$ requires commutativity.

2.3.6 Empty semigroup We specifically allow the **empty semigroup**, which consists of the empty set and the empty function from the empty set to itself. (Note that the cartesian product of the empty set with itself *is* the empty set.) This is not done in most of the non-category theory literature; it will become evident later (Section 8.5.2) why we should include the empty semigroup.

2.3.7 Definition An **identity element** e for a semigroup S is an element of S that satisfies the equation $se = es = s$ for all $s \in S$.

There can be at most one identity element in a semigroup.

2.3.8 Definition A **monoid** is a semigroup with an identity element. It is **commutative** if its binary operation is commutative.

It follows from the definition that a monoid is *not* allowed to be empty: it must contain an identity element. It also follows that we can extend the notation in 2.3.5 to 0 by defining x^0 to be the identity element of the monoid. The laws $s^k s^n = s^{k+n}$ and $(s^k)^n = s^{kn}$ then hold for all nonnegative k and n.

2.3.9 Examples One example of a semigroup is the set of positive integers with addition as the operation; this is a semigroup but not a monoid. If you include 0 you get a monoid.

The **Kleene closure** A^* of a set A is the set of strings (or lists) of finite length of elements of A. We write the lists in parentheses; for example (a, b, d, a) is an element of $\{a, b, c, d\}^*$. Some parts of the computer science literature call these strings instead of lists and write them this way: '*abda*'. A^* includes the empty list () and for each element $a \in A$ the list (a) of length one.

The operation of concatenation makes the Kleene closure a monoid $F(A)$, called the **free monoid** determined by A. The empty list is the identity element. We write concatenation as juxtaposition; thus

$$(a, b, d, a)(c, a, b) = (a, b, d, a, c, a, b)$$

Note that the underlying set of the free monoid is A^*, not A. In the literature, A is usually assumed finite, but the Kleene closure is defined for any set A. The elements of A^* are lists of *finite* length in any case. When A is nonempty, A^* is an infinite set.

The concept of freeness is a general concept applied to many kinds of structures. It is treated systematically in Chapter 9.

2.3.10 Definition A **submonoid** of a monoid M is a subset S of M with the properties:

SM–1 The identity element of M is in S.

SM–2 If $m, n \in S$ then $mn \in S$. (One says that S is **closed** under the operation.)

2.3.11 Examples The natural numbers with addition form a submonoid of the integers with addition. For another example, consider the integers with multiplication as the operation, so that 1 is the identity element. Again the natural numbers form a submonoid, and so does the set of *positive* natural numbers, since the product of two positive numbers is another one. Finally, the singleton set $\{0\}$ is a subset of the integers that is closed under multiplication, and it is a monoid, but it is not a submonoid of the integers on multiplication because it does not contain the identity element 1.

2.3.12 Monoid as category Any monoid M determines a category $C(M)$.

CM–1 $C(M)$ has one object, which we will denote $*$; $*$ can be chosen arbitrarily. A simple uniform choice is to take $* = M$.

CM–2 The arrows of $C(M)$ are the elements of M with $*$ as source and target.

CM–3 Composition is the binary operation on M.

(This construction is revisited in Section 4.4.)

Thus a category can be regarded as a generalized monoid, or a 'monoid with many objects'. This point of view has not been as fruitful in mathematics as the perception of a category as a generalized poset. It does have some applications in computing science; see [Barr and Wells, 1999], Chapter 12.

2.3.13 Remark Many categorists *define* a monoid to be a category with one object (compare 2.3.12) and a preordered set to be a category in which every hom set is either empty or a singleton (compare 2.3.1). This can be justified by the fact that the category of monoids and the category of one-object categories are 'equivalent' as defined in Section 4.4.

2.4 Categories of sets with structure

The typical use of categories has been to consider categories whose objects are sets with mathematical structure and whose arrows are functions that preserve that structure. The definition of category is an abstraction of basic properties of such systems. Typical examples have included categories whose objects are spaces of some type and whose arrows are continuous (or differentiable) functions between the spaces, and categories whose objects are algebraic structures of some specific type and whose arrows are homomorphisms between them.

In this section we describe various categories of sets with structure. The following section considers categories of semigroups and monoids.

Note the contrast with Section 2.3, where we discussed certain mathematical structures as categories. Here, we discuss categories whose *objects* are mathematical structures.

2.4.1 Definition The **category of graphs** has graphs as objects and homomorphisms of graphs (see 1.2.1) as arrows. It is denoted **GRF**. The category of graphs and homomorphisms between them is denoted **Grf**.

Let us check that the composite of graph homomorphisms is a graph homomorphism (identities are easy). Suppose $\phi : \mathscr{G} \to \mathscr{H}$ and $\psi : \mathscr{H} \to \mathscr{K}$ are graph homomorphisms, and suppose that $u : m \to n$ in $\mathscr{G}$. Then by definition $\phi_1(u) : \phi_0(m) \to \phi_0(n)$ in $\mathscr{H}$, and so by definition

$$\psi_1(\phi_1(u)) : \psi_0(\phi_0(m)) \to \psi_0(\phi_0(n)) \text{ in } \mathscr{K}$$

Hence $\psi \circ \phi$ is a graph homomorphism.

The identity homomorphism $\mathrm{id}_{\mathscr{G}}$ is the identity function for both nodes and arrows.

2.4.2 The category of posets If (S, α) and (T, β) are posets, a function $f : S \to T$ is **monotone** if whenever $x \alpha y$ in S, $f(x) \beta f(y)$ in T.

The identity function on a poset is clearly monotone, and the composite of two monotone functions is easily seen to be monotone, so that posets with monotone functions form a category. A variation on this is to consider only **strictly monotone** functions, which are functions f with the property that if $x \alpha y$ and $x \neq y$ then $f(x) \beta f(y)$ and $f(x) \neq f(y)$.

In 2.3.1, we saw how a single poset is a category. Now we are considering the *category* of *posets*.

We must give a few words of warning on terminology. The usual word in mathematical texts for what we have called 'monotone' is 'increasing' or 'monotonically increasing'. The word 'monotone' is used for a function that either preserves *or reverses* the order relation. That is, in mathematical texts a function $f : (X, \alpha) \to (T, \beta)$ is also called monotone if whenever $x \alpha y$ in S, $f(y) \beta f(x)$ in T.

2.5 Categories of algebraic structures

In this section, we discuss categories whose objects are semigroups or monoids. These are typical of categories of algebraic structures; we have concentrated on semigroups and monoids because transition systems naturally form monoids. The material in this section will come up primarily in examples later, and need not be thoroughly understood in order to read the rest of the notes.

2.5.1 Homomorphisms of semigroups and monoids If S and T are semigroups, a function $h : S \to T$ is a **homomorphism** if for all $s, s' \in S$, $h(ss') = h(s)h(s')$.

A **homomorphism of monoids** is a semigroup homomorphism between monoids that preserves the identity elements: if e is the identity element of the domain, $h(e)$ must be the identity element of the codomain.

2.5.2 Examples The identity function on any monoid is a monoid homomorphism. If M is a monoid and S is a submonoid (see 2.3.10), the inclusion function from S to M is a monoid homomorphism. Another example is the function that takes an even integer to 0 and an odd integer to 1. This is a monoid homomorphism from the monoid of integers on multiplication to the set $\{0, 1\}$ on multiplication.

It is easy to see that identity functions are homomorphisms and homomorphisms compose to give homomorphisms. Thus we have two categories: **Sem** is the category of semigroups and semigroup homomorphisms, and **Mon** is the category of monoids and monoid homomorphisms.

2.5.3 Example Let S be a semigroup with element s. Let $\mathbf{N}^+$ denote the semigroup of positive integers with addition as operation. There is a semigroup homomorphism $p : \mathbf{N}^+ \to S$ for which $p(k) = s^k$. That this is a homomorphism is just the statement $s^{k+n} = s^k s^n$ (see 2.3.5).

2.5.4 A *semigroup* homomorphism between monoids need not preserve the identities. An example of this involves the trivial monoid E with only one element e (which is perforce the identity element) and the monoid of all integers with multiplication as the operation, which is a monoid with identity 1. The function that takes the one element of E to 0 is a semigroup homomorphism that is not a monoid homomorphism. And, by the way, even though $\{0\}$ is a subsemigroup of the integers with multiplication and even though it is actually a monoid, it is *not* a submonoid.

2.5.5 Inverses of homomorphisms As an example of how to use the definition of homomorphism, we show that the inverse of a bijective semigroup homomorphism is also a semigroup homomorphism. Let $f : S \to T$ be a bijective semigroup homomorphism with inverse g. Let $t, t' \in T$. We have to show that $g(t)g(t') = g(tt')$. Since f is injective, it is sufficient to show that

$$f(g(t)g(t')) = f(g(tt'))$$

The right hand side is tt' because g is the inverse of f, and the left hand side is

$$f(g(t)g(t')) = f(g(t))f(g(t'))$$

because f is a homomorphism, but that is also tt' since g is the inverse of f.

This sort of theorem is true of other algebraic structures, such as monoids. It is not true for posets.

2.5.6 Isomorphisms of semigroups If a homomorphism of semigroups has an inverse that is a homomorphism (equivalently, as we just saw, if it is bijective), we say that the homomorphism is an **isomorphism**. In this case, the two semigroups in question have the same abstract structure and are said to be **isomorphic**. As we will see later, the property of possessing an inverse is taken to define the categorical notion of isomorphism (3.1.4).

It is important to understand that there may in general be many different isomorphisms between isomorphic semigroups. For example, there are two distinct isomorphisms between the monoid with underlying set $\{0, 1, 2, 3\}$ and addition (mod 4) as operation and the monoid with underlying set $\{1, 2, 3, 4\}$ and multiplication (mod 5) as operation.

We now discuss two important types of examples of monoid homomorphisms that will reappear later in the notes. The first example is a basic property of free monoids.

2.5.7 Kleene closure induces homomorphisms Let A and B denote sets, thought of as alphabets. Let $f : A \to B$ be any set function. We define $f^* : A^* \to B^*$ by $f^*\big((a_1, a_2, \ldots, a_k)\big) = (f(a_1), f(a_2), \ldots, f(a_k))$. In particular, $f^*() = ()$ and for any $a \in A$, $f^*(a) = f(a)$.

Then f^* is a homomorphism of monoids, a requirement that, in this case, means it preserves identity elements (by definition) and concatenation, which can be seen from the following calculation: Let $a = (a_1, a_2, \ldots, a_m)$ and $a' = (a'_1, a'_2, \ldots, a'_n)$ be lists in A^*. Concatenating them gives the list

$$aa' = (a_1, a_2, \ldots, a_m, a'_1, a'_2, \ldots, a'_n)$$

Then

$$\begin{aligned} f^*(a)f^*(a') &= f^*(a_1, a_2, \ldots, a_m)f^*(a'_1, a'_2, \ldots, a'_n) \\ &= (f(a_1), f(a_2), \ldots, f(a_m))(f(a'_1), f(a'_2), \ldots, f(a'_n)) \\ &= (f(a_1), f(a_2), \ldots, f(a_m), f(a'_1), f(a'_2), \ldots, f(a'_n)) \\ &= f^*(a_1, a_2, \ldots, a_m, a'_1, a'_2, \ldots, a'_n) \\ &= f^*(aa') \end{aligned}$$

Thus any set function between sets induces a monoid homomorphism between the corresponding free monoids. The reader may wish to verify that if f is an isomorphism then so is f^*.

The function f^* is called αf in [Backus, 1981a] and in modern functional languages is usually called map f or maplist f.

The other important example is a basic construction of number theory.

2.5.8 The remainder function The set $\mathbf{Z}$ of all integers forms a monoid with respect to either addition or multiplication. If k is any positive integer, the set $\mathbf{Z}_k = \{0, 1, \ldots, k-1\}$ of **remainders** of k is also a monoid with respect to addition or multiplication (mod k). Here are more precise definitions.

2.5.9 Definition Let k be a positive integer and n any integer. Then $n \bmod k$ is the unique integer $r \in \mathbf{Z}_k$ for which there is an integer q such that $n = qk + r$ and $0 \leq r < k$.

It is not difficult to see that there is indeed a unique integer r with these properties.
Define an operation '$+_k$' of addition (mod k) by requiring that

$$r +_k s = (r + s) \bmod k$$

The operation of addition of the contents of two registers in a microprocessor may be addition (mod k) for k some power of 2 (often complicated by the presence of sign bits).

2.5.10 Proposition *$(\mathbf{Z}_k, +_k)$ is a monoid with identity 0.*

We also have the following.

2.5.11 Proposition *The function $n \mapsto (n \bmod k)$ is a monoid homomorphism from $(\mathbf{Z}, +)$ to $(\mathbf{Z}_k, +_k)$.*

A similar definition and proposition can be given for multiplication.

2.6 Constructions on categories

If you are familiar with some branch of abstract algebra (for example the theory of semigroups, groups or rings) then you know that given two structures of a given type (e.g., two semigroups), you can construct a 'direct product' structure, defining the operations coordinatewise. Also, a structure may have substructures, which are subsets closed under the operations, and quotient structures, formed from equivalence classes modulo a congruence relation. Another construction that is possible in many cases is the formation of a 'free' structure of the given type for a given set.

All these constructions can be performed for categories. We will outline the constructions here, except for quotients, which are covered in [Barr and Wells, 1999], Section 3.5. We will also describe the construction of the slice category, which does not quite correspond to anything in abstract algebra (although it is akin to the adjunction of a constant to a logical theory). You do not need to be familiar with the constructions in other branches of abstract algebra, since they are all defined from scratch here.

2.6.1 Definition A **subcategory** $\mathscr{D}$ of a category $\mathscr{C}$ is a category for which:

S–1 All the objects of $\mathscr{D}$ are objects of $\mathscr{C}$ and all the arrows of $\mathscr{D}$ are arrows of $\mathscr{C}$ (in other words, $\mathscr{D}_0 \subseteq \mathscr{C}_0$ and $\mathscr{D}_1 \subseteq \mathscr{C}_1$).

S–2 The source and target of an arrow of $\mathscr{D}$ are the same as its source and target in $\mathscr{C}$ (in other words, the source and target maps for $\mathscr{D}$ are the restrictions of those for $\mathscr{C}$). It follows that for any objects A and B of $\mathscr{D}$, $\mathrm{Hom}_{\mathscr{D}}(A, B) \subseteq \mathrm{Hom}_{\mathscr{C}}(A, B)$.

S–3 If A is an object of $\mathscr{D}$ then its identity arrow id_A in $\mathscr{C}$ is in $\mathscr{D}$.

S–4 If $f : A \longrightarrow B$ and $g : B \longrightarrow C$ in $\mathscr{D}$, then the composite (in $\mathscr{C}$) $g \circ f$ is in $\mathscr{D}$ and is the composite in $\mathscr{D}$.

2.6.2 Examples As an example, the category **Fin** of finite sets and all functions between them is a subcategory of **Set**, and in turn **Set** is a subcategory of the category of sets and *partial* functions between sets (see 2.1.12 and 2.1.13). These examples illustrate two phenomena:

(i) If A and B are finite sets, then $\mathrm{Hom}_{\mathbf{Fin}}(A, B) = \mathrm{Hom}_{\mathbf{Set}}(A, B)$. In other words, every arrow of **Set** between objects of **Fin** is an arrow of **Fin**.

(ii) The category of sets and the category of sets and partial functions, on the other hand, have exactly the same *objects*. The phenomenon of (i) does not occur here: there are generally *many* more partial functions between two sets than there are full functions.

Example (i) motivates the following definition.

2.6.3 Definition If $\mathscr{D}$ is a subcategory of $\mathscr{C}$ and for every pair of objects A, B of $\mathscr{D}$, $\mathrm{Hom}_{\mathscr{D}}(A,B) = \mathrm{Hom}_{\mathscr{C}}(A,B)$, then $\mathscr{D}$ is a **full subcategory** of $\mathscr{C}$.

Thus **Fin** is a full subcategory of **Set** but **Set** is not a full subcategory of the category of sets and partial functions.

Example 2.6.2(ii) also motivates a (less useful) definition, as follows.

2.6.4 Definition If $\mathscr{D}$ is a subcategory of $\mathscr{C}$ with the same objects, then $\mathscr{D}$ is a **wide** subcategory of $\mathscr{C}$.

Thus in the case of a wide subcategory, only the arrows are different from those of the larger category. In 3.1.8 we provide an improvement on this concept.

As an example, **Set** is a wide subcategory of the category **Pfn** of sets and partial functions.

2.6.5 Example Among all the objects of the category of semigroups are the monoids, and among all the semigroup homomorphisms between two monoids are those that preserve the identity. Thus the category of monoids is a subcategory of the category of semigroups that is neither wide nor full (for the latter, see 2.5.4).

As it stands, being a subcategory requires the objects and arrows of the subcategory to be identical with some of the objects and arrows of the category containing it. This requires an uncategorical emphasis on what something *is* instead of on the specification it satisfies. We will return to this example in 3.3.14 and again in 4.1.8.

2.6.6 The product of categories If $\mathscr{C}$ and $\mathscr{D}$ are categories, the **product** $\mathscr{C} \times \mathscr{D}$ is the category whose objects are all ordered pairs (C,D) with C an object of $\mathscr{C}$ and D an object of $\mathscr{D}$, and in which an arrow $(f,g) : (C,D) \to (C',D')$ is a pair of arrows $f : C \to C'$ in $\mathscr{C}$ and $g : D \to D'$ in $\mathscr{D}$. The identity of (C,D) is $(\mathrm{id}_C, \mathrm{id}_D)$. If $(f',g') : (C',D') \to (C'',D'')$ is another arrow, then the composite is defined by

$$(f',g') \circ (f,g) = (f' \circ f, g' \circ g) : (C,D) \to (C'',D'')$$

2.6.7 The dual of a category Given any category $\mathscr{C}$, you can construct another category denoted $\mathscr{C}^{\mathrm{op}}$ by reversing all the arrows. The **dual** or **opposite** $\mathscr{C}^{\mathrm{op}}$ of a category $\mathscr{C}$ is defined by:

D–1 The objects and arrows of $\mathscr{C}^{\mathrm{op}}$ are the objects and arrows of $\mathscr{C}$.

D–2 If $f : A \to B$ in $\mathscr{C}$, then $f : B \to A$ in $\mathscr{C}^{\mathrm{op}}$.

D–3 If $h = g \circ f$ in $\mathscr{C}$, then $h = f \circ g$ in $\mathscr{C}^{\mathrm{op}}$.

The meaning of D–2 is that source and target have been reversed. It is easy to see that the identity arrows have to be the same in the two categories $\mathscr{C}$ and $\mathscr{C}^{\mathrm{op}}$ and that C–1 through C–4 of Section 2.1 hold, so that $\mathscr{C}^{\mathrm{op}}$ is a category.

2.6.8 Example If M is a monoid, then the opposite of the category $C(M)$ is the category determined by a monoid M^{op}; if $xy = z$ in M, then $yx = z$ in M^{op}. (Hence if M is commutative then $C(M)$ is its own dual. Similar remarks may be made about the opposite of the category $C(P)$ determined by a poset P. The opposite of the poset $(\mathbf{Z}, \leq)$, for example, is $(\mathbf{Z}, \geq)$.

2.6.9 Both the construction of the product of two categories and the construction of the dual of a category are purely formal constructions. Even though the original categories may have, for example, structure-preserving functions of some kind as arrows, the arrows in the product category are simply pairs of arrows of the original categories.

Consider **Set**, for example. Let A be the set of letters of the English alphabet. The function $v : A \to \{0,1\}$ that takes consonants to 0 and vowels to 1 is an arrow of **Set**. Then the arrow $(\mathrm{id}_A, v) : (A, A) \to (A, \{0,1\})$ of **Set** $\times$ **Set** is not a function, not even a function of two variables; it is merely the arrow of a product category and as such is an ordered pair of functions.

A similar remark applies to duals. In $\mathbf{Set}^{\mathrm{op}}$, v is an arrow from $\{0,1\}$ to A. And that is all it is. It is in particular *not* a function from $\{0,1\}$ to A.

Nevertheless, it is possible in some cases to prove that the dual of a familiar category is essentially the same as some other familiar category. One such category is **Fin**, which is equivalent to the opposite of the category of finite Boolean algebras.

The product of categories is a formal way to make constructions dependent on more than one variable. The major use we make of the concept of dual is that many of the definitions we make have another meaning when applied to the dual of a category that is often of independent interest. The phrase **dual concept** or **dual notion** is often used to refer to a concept or notion applied in the dual category.

2.6.10 Slice categories If $\mathscr{C}$ is a category and A any object of $\mathscr{C}$, the **slice category** $\mathscr{C}/A$ is described this way:

SC–1 An object of $\mathscr{C}/A$ is an arrow $f : C \to A$ of $\mathscr{C}$ for some object C.

SC–2 An arrow of $\mathscr{C}/A$ from $f : C \to A$ to $f' : C' \to A$ is an arrow $h : C \to C'$ with the property that $f = f' \circ h$.

SC–3 The composite of $h : f \to f'$ and $h' : f' \to f''$ is $h' \circ h$.

It is necessary to show that $h' \circ h$, as defined in SC–2, satisfies the requirements of being an arrow from f to f''. Let $h : f \to f'$ and $h' : f' \to f''$. This means $f' \circ h = f$ and $f'' \circ h' = f'$. To show that $h' \circ h : f \to f''$ is an arrow of $\mathscr{C}/A$, we must show that $f'' \circ (h' \circ h) = f$. That follows from this calculation:

$$f'' \circ (h' \circ h) = (f'' \circ h') \circ h = f' \circ h = f$$

The usual notation for arrows in $\mathscr{C}/A$ is deficient: the same arrow h can satisfy $f = f' \circ h$ and $g = g' \circ h$ with $f \neq g$ or $f' \neq g'$ (or both). Then $h : f \to f'$ and $h : g \to g'$ are *different* arrows of $\mathscr{C}/A$.

2.6.11 Example Let (P, α) be a poset and let $C(P)$ be the corresponding category as in 2.3.1. For an element $x \in P$, the slice category $C(P)/x$ is the category corresponding to the set of elements greater than or equal to x. The dual notion of coslice gives the set of elements less than or equal to a given element.

2.6.12 The importance of slice categories comes in part with their connection with indexing. An **S-indexed set** is a set X together with a function $\tau : X \to S$. If $x \in X$ and $\tau(x) = s$ then we say x is of **type** s, and we also refer to X as a **typed set**.

The terminology 'S-indexed set' is that used by category theorists. Many mathematicians would cast the discussion in terms of the collection $\{\tau^{-1}(s) \mid s \in S\}$ of subsets of X, which would be called a **family of sets indexed by** S.

2.6.13 Example The set $G = G_0 \cup G_1$ of objects and arrows of a graph $\mathscr{G}$ is an example of a typed set, typed by the function $\tau : G \to \{0,1\}$ that takes a node to 0 and an arrow to 1. Note that this depends on the fact that a node is not an arrow: G_0 and G_1 are disjoint.

2.6.14 Indexed functions A function from a set X typed by S to a set X' typed by the same set S that preserves the typing (takes an element of type s to an element of type s) is exactly an arrow of the slice category $\mathbf{Set}/S$. Such a function is called an **indexed function** or **typed function**. It has been fruitful for category theorists to pursue this analogy by thinking of objects of any slice category $\mathscr{C}/A$ as objects of $\mathscr{C}$ indexed by A.

2.6.15 Example A graph homomorphism $f : \mathscr{G} \longrightarrow \mathscr{H}$ corresponds to a typed function according to the construction in Example 2.6.13. However, there are typed functions between graphs that are not graph homomorphisms, for example the function from the graph (1.1), page 1, to the graph (1.3), page 2, defined by

$$1 \mapsto 1, 2 \mapsto \mathsf{n}, a \mapsto 0, b \mapsto 0, c \mapsto \mathsf{succ}$$

This is not a graph homomorphism because it does not preserve source and target.

2.6.16 The free category generated by a graph For any given graph $\mathscr{G}$ there is a category $F(\mathscr{G})$ whose objects are the nodes of $\mathscr{G}$ and whose arrows are the paths in $\mathscr{G}$. Composition is defined by the formula

$$(f_1, f_2, \ldots, f_k) \circ (f_{k+1}, \ldots, f_n) = (f_1, f_2, \ldots, f_n)$$

This composition is associative, and for each object A, id_A is the empty path from A to A. The category $F(\mathscr{G})$ is called the **free category generated by the graph** $\mathscr{G}$. It is also called the **path category** of $\mathscr{G}$.

2.6.17 Examples The free category generated by the graph with one node and no arrows is the category with one object and only the identity arrow, which is the empty path. The free category generated by the graph with one node and one loop on the node is the free monoid with one generator (Kleene closure of a one-letter alphabet); this is isomorphic with the nonnegative integers with + as operation.

The free category generated by the graph in Example 1.1.5 has the following arrows

(a) An arrow $\mathrm{id}_1 : 1 \longrightarrow 1$.

(b) For each nonnegative integer k, the arrow $\mathsf{succ}^k : \mathsf{n} \longrightarrow \mathsf{n}$. This is the path $(\mathsf{succ}, \mathsf{succ}, \ldots, \mathsf{succ})$ (k occurrences of succ). This includes $k = 0$ which gives id_{n}.

(c) For each nonnegative integer k, the arrow $\mathsf{succ}^k \circ 0 : 1 \longrightarrow \mathsf{n}$. Here $k = 0$ gives $0 : 1 \longrightarrow \mathsf{n}$.

Composition obeys the rule $\mathsf{succ}^k \circ \mathsf{succ}^m = \mathsf{succ}^{k+m}$.

2.6.18 It is useful to regard the free category generated by *any* graph as analogous to Kleene closure (free monoid) generated by a set (as in 2.3.9). The paths in the free category correspond to the strings in the Kleene closure. The difference is that you can concatenate any symbols together to get a string, but arrows can be strung together only head to tail, thus taking into account the typing.

In 9.2.3 we give a precise technical meaning to the word 'free'.

3. Properties of objects and arrows in a category

The data in the definition of category can be used to define properties that the objects and arrows of the category may have. A property that is defined strictly in terms of the role the object or arrow has in the category, rather than in terms of what it really is in any sense, is called a **categorical definition**. Such definitions are *abstract* in the sense that a property a thing can have is defined entirely in terms of the external interactions of that thing with other entities.

The examples of categorical definitions in this chapter are of several simple concepts that can be expressed directly in terms of the data used in the definition of category. Other concepts, such as limit, naturality and adjunction, require deeper ideas that will be the subject of succeeding chapters.

3.1 Isomorphisms

In general, the word 'isomorphic' is used in a mathematical context to mean indistinguishable in form. We have already used it in this way in 2.5.6. It turns out that it is possible to translate this into categorical language in a completely satisfactory way. To do this, we first need the concept of inverse.

3.1.1 Definition Suppose $f : A \longrightarrow B$ and $g : B \longrightarrow A$ are arrows in a category for which $f \circ g$ is the identity arrow of B and $g \circ f$ is the identity arrow of A. Then g is an **inverse** to f, and, of course, f is an inverse to g.

3.1.2 As an example of how to use the definition of inverse, we show that if $f : A \longrightarrow B$ has an inverse, it has only one. Suppose $g : B \longrightarrow A$ and $h : B \longrightarrow A$ have the properties that $g \circ f = h \circ f = \mathrm{id}_A$ and $f \circ g = f \circ h = \mathrm{id}_B$. Then

$$g = g \circ \mathrm{id}_B = g \circ (f \circ h) = (g \circ f) \circ h = \mathrm{id}_A \circ h = h$$

Note that this does not use the full power of the hypothesis.

From this uniqueness, we can conclude that if $f : A \longrightarrow A$ has an inverse, then $(f^{-1})^{-1} = f$. Proof: All four of the following equations are true by definition of inverse:

(i) $f^{-1} \circ f = \mathrm{id}_A$.
(ii) $f \circ f^{-1} = \mathrm{id}_A$.
(iii) $f^{-1} \circ (f^{-1})^{-1} = \mathrm{id}_A$.
(iv) $(f^{-1})^{-1} \circ f^{-1} = \mathrm{id}_A$.

It follows that both f and $(f^{-1})^{-1}$ are arrows g such that $f^{-1} \circ g = \mathrm{id}_A$ and $g \circ f^{-1} = \mathrm{id}_A$. Thus $f = (f^{-1})^{-1}$ by uniqueness of the inverse of f^{-1}.

3.1.3 Proposition *If $f : A \longrightarrow B$ and $g : B \longrightarrow C$ are isomorphisms in a category with inverses $f^{-1} : B \longrightarrow A$ and $g^{-1} : C \longrightarrow B$, then $g \circ f$ is an isomorphism with inverse $f^{-1} \circ g^{-1}$.*

The proof is omitted. This Proposition is sometimes called the 'Shoe–Sock Theorem': to undo the act of putting on your socks, then your shoes, you have to take off your *shoes*, then your *socks*.)

3.1.4 Definition Suppose that $\mathscr{C}$ is a category and that A and B are two objects of $\mathscr{C}$. An arrow $f : A \longrightarrow B$ is said to be an **isomorphism** if it has an inverse. In that case, we say that A is **isomorphic to** B, written $A \cong B$.

In a monoid, an element which is an isomorphism in the corresponding category is usually called **invertible**.

3.1.5 Definition An arrow $f : A \longrightarrow A$ in a category (with the source and target the same) is called an **endomorphism**. If it is invertible, it is called an **automorphism**.

3.1.6 Examples Any identity arrow in any category is an isomorphism (hence an automorphism). In the category determined by a partially ordered set, the *only* isomorphisms are the identity arrows. If, in the category determined by a monoid, every arrow is an isomorphism the monoid is called a **group**. Because of this, a category in which every arrow is an isomorphism is called a **groupoid**.

In the category of semigroups and semigroup homomorphisms, and likewise in the category of monoids and monoid homomorphisms, the isomorphisms are exactly the bijective homomorphisms. On the other hand, in the category of posets and monotone maps, there are bijective homomorphisms that have no inverse.

3.1.7 Definition A property that objects of a category may have is **preserved by isomorphisms** if for any object A with the property, any object isomorphic to A must also have the property.

3.1.8 From the categorist's point of view there is no reason to distinguish between two isomorphic objects in a category, since the interesting fact about a mathematical object is the way it relates to other mathematical objects and two isomorphic objects relate to other objects in the same way. For this reason, the concept of wide category (Definition 2.6.4) is not in the spirit of category theory. What really should matter is whether the subcategory contains an isomorphic copy of every object in the big category. This motivates the following definition.

3.1.9 Definition A subcategory $\mathscr{D}$ of a category $\mathscr{C}$ is said to be a **representative subcategory** if every object of $\mathscr{C}$ is isomorphic to some object of $\mathscr{D}$.

3.1.10 Example Let $\mathscr{D}$ be the category whose objects are the empty set and all sets of the form $\{1, 2, \ldots, n\}$ for some positive integer n and whose arrows are all the functions between them. Then $\mathscr{D}$ is a representative subcategory of **Fin** (Definition 2.1.12), since there is a bijection from any nonempty finite set to some set of the form $\{1, 2, \ldots, n\}$. Note that $\mathscr{D}$ is also full in **Fin**.

3.2 Terminal and initial objects

An object T of a category $\mathscr{C}$ is called **terminal** if there is *exactly* one arrow $A \longrightarrow T$ for each object A of $\mathscr{C}$. We usually denote the terminal object by 1 and the unique arrow $A \longrightarrow 1$ by $\langle\rangle$.

The dual notion (see 2.6.7), an object of a category that has a unique arrow *to* each object (including itself), is called an **initial object** and is often denoted 0.

3.2.1 Examples In the category of sets, the empty set is initial and any one-element set is terminal. Thus the category of sets has a *unique* initial object but many terminal objects. The one-element monoid is both initial and terminal in the category of monoids. In the category determined by a poset, an object is initial if and only if it is an absolute minimum for the poset, and it is terminal if and only if it is an absolute maximum. Since there is no largest or smallest whole number, the category determined by the set of integers with its natural order (there is an arrow from m to n if and only if $m \leq n$) gives an example of a category without initial or terminal object.

In the category of semigroups, the empty semigroup (see 2.3.6) is the initial object and any one-element semigroup is a terminal object. On the other hand, the category of *nonempty* semigroups does not have an initial object. Warning: To prove this, it is not enough to say that the initial object in the category of semigroups is the empty semigroup and that semigroup is missing here! You have to show that no other object can be the initial object in the smaller category. One way to do this is to let U be the semigroup with two elements 1 and e with $1 \cdot e = e \cdot 1 = e$, $1 \cdot 1 = 1$ and $e \cdot e = e$. Then any nonempty semigroup S has *two* homomorphisms to U: the constant function taking everything to 1 and the one taking everything to e. Thus no nonempty semigroup S can be the initial object.

In the category of graphs and graph homomorphisms, the graph with one node and one arrow is the terminal object.

3.2.2 Proposition *Any two terminal (respectively initial) objects in a category are isomorphic.*

Proof. Suppose T and T' are terminal objects. Since T is terminal, there is an arrow $f : T' \to T$. Similarly, there is an arrow $g : T \to T'$. The arrow $f \circ g : T \to T$ is an arrow with target T. Since T is a terminal object of the category, there can be only one arrow from T to T. Thus it must be that $f \circ g$ is the identity of T. An analogous proof shows that $g \circ f$ is the identity of T'. □

3.2.3 Constants In **Set**, an element x of a set A is the image of a function from a singleton set to A that takes the unique element of the singleton to x. Thus if we pick a specific singleton $\{*\}$ and call it 1, the elements of the set A are in one to one correspondence with $\mathrm{Hom}(1, A)$, which is the set of functions from the terminal object to A. Moreover, if $f : A \to B$ is a set function and x is an element of A determining the function $x : 1 \to A$, then the element $f(x)$ of B is essentially the same thing as the composite $f \circ x : 1 \to B$. Because of this, the categorist typically thinks of an element $x \in A$ as *being* the constant $x : 1 \to A$.

An arrow $1 \to A$ in a category, where 1 is the terminal object, is called a **constant of type** A. Thus each element of a set is a constant in **Set**. On the other hand, each monoid M has just one constant $1 \to M$ in the category of monoids, since monoid homomorphisms must preserve the identity.

The more common name in the categorical literature for a constant is **global element** of A, a name that comes from sheaf theory (see Section 11.4).

> A terminal object is an object with exactly one arrow $\langle\rangle : A \to 1$ to it from each object A. So the arrows *to* 1 are not interesting. Global elements are arrows *from* the terminal object. There may be none or many, so *they* are interesting.

3.2.4 If 1 and $1'$ are two terminal objects in a category and $x : 1 \to A$ and $x' : 1' \to A$ are two constants with the property that $x' \circ \langle\rangle = x$ (where $\langle\rangle$ is the unique isomorphism from 1 to $1'$), then we regard x and x' as the *same* constant. Think about this comment as it applies to elements in the category of sets, with two different choices of terminal object, and you will see why.

3.3 Monomorphisms and subobjects

3.3.1 Monomorphisms A function $f : A \to B$ in **Set** is injective if for any $x, y \in A$, if $x \neq y$, then $f(x) \neq f(y)$. A monomorphism is a particular type of arrow in a category which generalizes the concept of injective function; in particular, a monomorphism in the category of sets is exactly an injective function. If f is an arrow in an arbitrary category, we use the same definition, except for one change required because the concept of 'element' no longer makes sense.

3.3.2 Definition $f : A \to B$ is a **monomorphism** if for any object T of the category and any arrows $x, y : T \to A$, if $x \neq y$, then $f \circ x \neq f \circ y$.

We often write $f : A \rightarrowtail B$ to indicate that f is a monomorphism and say that f is **monic** or that f is **mono**.

In Definition 3.3.2 and many like it, what replaces the concept of element of A is an arbitrary arrow into A. In this context, an arbitrary arrow $a : T \to A$ is called a **variable element** of A, parametrized by T. When a is treated as a variable element and f has source A, one may write $f(a)$ for $f \circ a$. Using this notation, f is a monomorphism if for any variable elements $x, y : T \to A$, if $x \neq y$, then $f(x) \neq f(y)$.

The following theorem validates the claim that 'monomorphism' is the categorical version of 'injective'.

3.3.3 Theorem *In the category of sets, a function is injective if and only if it is a monomorphism.*

Proof. Suppose $f : A \to B$ is injective, and let $a, a' : T \to A$ be variable elements of A. If $a \neq a'$ then there is an (ordinary) element $t \in T$ for which $a(t) \neq a'(t)$. Then $f(a(t)) \neq f(a'(t))$, so $f \circ a \neq f \circ a'$. Hence f is monic.

Conversely, suppose f is monic. Since global elements (see 3.2.3) are elements, this says that for any global elements $x, y : 1 \to A$ with $x \neq y$, $f \circ x \neq f \circ y$, i.e., $f(x) \neq f(y)$, which means that f is injective. □

3.3.4 Remark An arrow that is a monomorphism in a category is a monomorphism in any subcategory it happens to be in. However, an arrow can be a monomorphism in a subcategory of a category $\mathscr{C}$ without being a monomorphism in $\mathscr{C}$.

3.3.5 Examples In most familiar categories of sets with structure and functions that preserve structure, the monomorphisms are exactly the injective functions. In particular, the monomorphisms in **Mon** are the injective homomorphisms (proved in 3.3.6 below). This is evidence that Definition 3.3.2 is the correct categorical definition generalizing the set-theoretic concept of injectivity.

In the category determined by a poset, every arrow is monic. A monic element of the category determined by a monoid is generally called **left cancellable**.

An isomorphism in any category is a monomorphism. For suppose f is an isomorphism and $f \circ x = f \circ y$. This calculation shows that $x = y$:

$$x = f^{-1} \circ f \circ x = f^{-1} \circ f \circ y = y$$

3.3.6 We now show that a monomorphism in the category of monoids is an injective homomorphism, and conversely.

Let $f : M \longrightarrow M'$ be a monoid homomorphism. Suppose it is injective. Let $g, h : V \longrightarrow M$ be homomorphisms for which $f \circ g = f \circ h$. For any $v \in V$, $f(g(v)) = f(h(v))$, so $g(v) = h(v)$ since f is injective. Hence $g = h$. It follows that f is a monomorphism. Essentially the same proof works in other categories of structures and structure-preserving maps – if the map is injective it is a monomorphism for the same reason as in **Set**.

However, the converse definitely does not work that way. The proof for **Set** in Theorem 3.3.3 above uses distinct global elements x and y, but a monoid need not have distinct global elements. For example, let $\mathbf{N}$ denote the monoid of nonnegative integers with addition as operation. Then the only global element of $\mathbf{N}$ on addition is 0. So we have to work harder to get a proof.

Suppose f is a monomorphism. Let $x, y \in M$ be distinct elements. Let $p_x : \mathbf{N} \longrightarrow M$ take k to x^k and similarly define p_y; p_x and p_y are homomorphisms since for all x, $x^{k+n} = x^k x^n$ (see 2.3.5 and the discussion after Definition 3.3.2). Since $x \neq y$, p_x and p_y are *distinct* homomorphisms. If $f(x) = f(y)$ then for all positive integers k,

$$f(p_x(k)) = f(x^k) = f(x)^k = f(y)^k = f(y^k) = f(p_y(k))$$

so that $f \circ p_x = f \circ p_y$ which would mean that f is not a monomorphism. Thus we must have $f(x) \neq f(y)$ so that f is injective.

The trick in the preceding paragraph was to find an object ($\mathbf{N}$ here) that allows one to distinguish elements of the arbitrary monoid M. In **Set**, the corresponding object was the terminal object, but that does not work for **Mon**: each monoid has exactly one global element because a map from the one-element monoid must have the identity element as value.

We now state two propositions that give some elementary properties of monomorphisms.

3.3.7 Proposition *Suppose $f : A \longrightarrow B$ and $g : B \longrightarrow C$ in a category $\mathscr{C}$. Then*

(a) *If f and g are monomorphisms, so is $g \circ f$.*
(b) *If $g \circ f$ is a monomorphism, so is f.*

Proof. We prove the second statement and leave the first to you. Suppose $g \circ f$ is a monomorphism. To show that f is a monomorphism, assume $f \circ x = f \circ y$ for some arrows $x, y : C \longrightarrow A$. Then

$$(g \circ f) \circ x = g \circ (f \circ x) = g \circ (f \circ y) = (g \circ f) \circ y$$

so, since $g \circ f$ is a monomorphism, $x = y$. □

3.3.8 Proposition *Let $m : C \longrightarrow 0$ be a monomorphism into an initial object. Then m is an isomorphism.*

Proof. Let $i : 0 \longrightarrow C$ be the unique arrow given by definition of initial object. Then $m \circ i$ and id_0 are both arrows from 0 to 0 and so must be the same. It remains to show that $i \circ m = \mathrm{id}_C$. This follows from the fact that $m \circ i \circ m = m \circ \mathrm{id}_C$ and that m is a monomorphism. □

3.3.9 Subobjects The concept of subobject is intended to generalize the concept of subset of a set, submonoid of a monoid, subcategory of a category, and so on. This idea cannot be translated exactly into categorical terms, since the usual concept of subset violates the strict typing rules of category theory: to go from a subset to a set requires a change of type, so there is no feasible way to say that the same element x is in both a set and a subset of the set.

Because of this, any categorical definition of subobject will not give exactly the concept of subset when applied to the category of sets. However, the usual definition of subobject (which we give here in Definition 3.3.12) produces, in **Set**, a concept that is naturally equivalent to the concept of subset in a strong sense that we will describe in 3.3.13. The definition, when applied to sets, defines subset in terms of the inclusion function.

3.3.10 We need a preliminary idea. If $f : A \to B$ is an arrow in a category, and for some arrow $g : C \to B$ there is an arrow $h : A \to C$ for which $f = g \circ h$, we say f **factors through** g. This is because the equation $g \circ h = f$ can be solved for h.

> The use of the word 'factor' shows the explicit intention of categorists to work with functions in an algebraic manner: a category is an algebra of functions.

Suppose $f_0 : C_0 \to C$ and $f_1 : C_1 \to C$ are monomorphisms in a category. Let us say that $f_0 \sim f_1$ if each factors through the other.

3.3.11 Proposition *Let $f_0 \sim f_1$. Then the factors implied by the definition of $\sim$ are unique and are inverse isomorphisms. Moreover, the relation $\sim$ is an equivalence relation on the collection of arrows with target C.*

Proof. The definition implies the existence of arrows $g : C_0 \to C_1$ and $h : C_1 \to C_0$ such that $f_1 \circ g = f_0$ and $f_0 \circ h = f_1$. The arrows g and h are unique because f_0 and f_1 are monomorphisms. Moreover, $f_1 \circ g \circ h = f_0 \circ h = f_1 = f_1 \circ \mathrm{id}$; since f_1 is a monomorphism, we conclude that $g \circ h = \mathrm{id}$. Similarly, $h \circ g = \mathrm{id}$.

That $\sim$ is reflexive follows by taking the factor to be the identity arrow, and it is symmetric by definition. For transitivity, you get the required factor by composing the given factors; we leave the details to you. □

3.3.12 Definition In a category $\mathscr{C}$, a **subobject** of an object C is an equivalence class of monomorphisms under $\sim$. The subobject is a **proper subobject** if it does not contain id_C.

Observe that it follows immediately from Proposition 3.3.8 that an initial object in a category has no proper subobjects.

3.3.13 Subobjects in the category of sets In **Set**, a monomorphism is an injection, so a subobject is an equivalence class of injections. The following sequence of statements are each easy to prove and together form a precise description of the connection between subobjects and subsets in the category of sets. Similar remarks can be made about other categories of sets with structure, such as semigroups, monoids or posets.

In these statements, S is a set.

(a) Let $\mathscr{O}$ be a subobject of S.

  (i) Any two injections $m : A \to S$ and $n : B \to S$ in $\mathscr{O}$ have the same image; call the image I.

  (ii) The inclusion $i : I \to S$ is equivalent to any injection in $\mathscr{O}$, hence is an element of $\mathscr{O}$.

  (iii) If $j : J \to S$ is an inclusion of a subset J into S that is in $\mathscr{O}$, then $I = J$ and $i = j$.

  (iv) Hence every subobject of S contains exactly one inclusion of a subset of S into S, and that subset is the image of any element of $\mathscr{O}$.

(b) Let $i : T \to S$ be the inclusion of a subset T of S into S.

  (i) Since i is injective, it is an element of a subobject of S.

  (ii) Since the subobjects are equivalence classes of an equivalence relation, they are disjoint, so i is not in two subobjects.

(iii) Hence the subsets of S with their inclusion maps form a complete set of class representatives for the subobjects of S.

Thus subobjects, given by a categorical definition, are not the same as subsets, but each subset determines and is determined by a unique subobject.

Because of this close relationship, one frequently says, of objects A and B in a category, 'Let A be a subobject of B', meaning that one has in mind a certain equivalence class of monomorphisms that in particular contains a monomorphism $A \rightarrowtail B$. You should be aware that there may be many other monomorphisms from A to B that are not in the equivalence class, just as from any subset A of a set B there are generally many injective functions from A to B other than the inclusion.

3.3.14 As a consequence of the properties of the subobject construction, categorists take a different attitude toward substructures such as subsets and submonoids, as compared to many other mathematicians. For them, A is a subobject or substructure of B if there is a monomorphism from A to B, and the subobject is the equivalence class determined by that monomorphism. For example, let $\mathbf{Z}$ denote the set of integers and $\mathbf{R}$ the set of real numbers. In calculus classes, $\mathbf{Z}$ is a subset of $\mathbf{R}$; an integer actually is a real number. For the categorist, it suffices that there be a monic (injective) map from $\mathbf{Z}$ to $\mathbf{R}$.

That monic map is a kind of *type conversion*. (See [Reynolds, 1980] for a more general view.) An integer need not actually be thought of as a real number, but there is a standard or canonical way (translate this statement as 'a monic map') to regard an integer as a real number. This mapping is regarded by the categorist as an inclusion, even though in fact it may change what the integer really is.

In a computer language, converting an integer to a real may increase the storage allotted to it and change its representation. Something similar happens in many approaches to the foundations of mathematics, where real numbers are constructed from integers by a complicated process (Dedekind cuts or Cauchy sequences), which results in an embedding of the integers in the real numbers. Just as for computer languages, this embedding changes the form of an integer: instead of whatever it was before, it is now a Dedekind cut (or Cauchy sequence).

In traditional texts on foundations, this construction had to be modified to replace the image of each integer by the actual integer, so that the integers were actually inside the real numbers. From the categorical point of view, this is an unnecessary complication. This change results in replacing a monomorphism $\mathbf{Z} \rightarrow \mathbf{R}$ by an equivalent monomorphism (one that determines the same subobject). From an *operational* point of view, the integers behave the same way whether this change is made or not.

3.3.15 Categories and typing In category theory, the inclusion map is usually made explicit. From the computing science point of view, category theory is a very strongly typed language, more strongly typed than any computer language. For example, the strict categorist will refer explicitly to the inclusion map from the nonzero real numbers to the set of all real numbers when talking of division. In a computer language this would correspond to having two different types, `REAL` and `NONZERO_REAL`, set up in such a way that you can divide a `REAL` only by a `NONZERO_REAL`. To multiply a `REAL` by a `NONZERO_REAL`, the strong typing would require you to convert the latter to a `REAL` first.

To be sure, categorists themselves are not always so strict; but when they are not strict they are aware of it. Nor is this discussion meant to imply that computer languages should have such strict typing: rather, the intention is to illustrate the way category theory handles types.)

3.4 Other types of arrow

3.4.1 Epimorphisms Epimorphisms in a category are the same as monomorphisms in the dual category. So $f : S \rightarrow T$ is an epimorphism if for any arrows $g, h : T \rightarrow X$, $g \circ f = h \circ f$ implies $g = h$. An epimorphism is said to be **epic** or an **epi**, and may be denoted with a double-headed arrow, as in $f : S \twoheadrightarrow T$.

3.4.2 Proposition *A set function is an epimorphism in* **Set** *if and only if it is surjective.*

Proof. Suppose $f : S \to T$ is surjective, and $g, h : T \to X$ are two functions. If $g \neq h$, then there is some particular element $t \in T$ for which $g(t) \neq h(t)$. Since f is surjective, there is an element $s \in S$ for which $f(s) = t$. Then $g(f(s)) \neq h(f(s))$, so that $g \circ f \neq h \circ f$.

Conversely, suppose f is *not* surjective. Then there is some $t \in T$ for which there is no $s \in S$ such that $f(s) = t$. Now define two functions $g : T \to \{0, 1\}$ and $h : T \to \{0, 1\}$ as follows:

(i) $g(x) = h(x) = 0$ for all x in T except t.

(ii) $g(t) = 0$.

(iii) $h(t) = 1$.

Then $g \neq h$ but $g \circ f = h \circ f$, so f is not an epimorphism. □

3.4.3 In contrast to the situation with monomorphisms, epimorphisms in categories of sets with structure are commonly not surjective. For example the nonnegative integers and the integers are both monoids under addition, and the inclusion function i is a homomorphism which is certainly not surjective. However, it is an epimorphism. (Note that surjective homomorphisms in the category of monoids are always epimorphisms.)

Here is the proof: any homomorphism h whose domain is the integers is determined completely by its value $h(1)$. For positive m, $m = 1 + 1 + \cdots + 1$, so

$$h(m) = h(1 + 1 + \cdots + 1) = h(1)h(1) \cdots h(1)$$

where we write the operation in the codomain as juxtaposition. Also, $h(-1)$ is the inverse of $h(1)$, since

$$h(1)h(-1) = h(-1)h(1) = h(-1 + 1) = h(0)$$

which must be the identity of the codomain. Since an element of a monoid can have only one inverse, this means $h(-1)$ is uniquely determined by $h(1)$. Then since every negative integer is a sum of -1's, the value of h at every negative integer is also determined by its value at 1.

Now suppose that g and h are two homomorphisms from the monoid of integers into the same codomain. Then g and h are both determined by their value at 1. Since 1 is a positive integer, this means that if $g \circ i = h \circ i$, then $g = h$. Thus i is an epimorphism.

3.4.4 Proposition *Let $f : A \to B$ and $g : B \to C$. Then*

(a) *If f and g are epimorphisms, so is $g \circ f$.*

(b) *If $g \circ f$ is an epimorphism, so is g.*

Proof. This is the dual of Proposition 3.3.7.

3.4.5 Proposition *A homomorphism $f : \mathscr{G} \to \mathscr{H}$ of graphs is an epimorphism if and only if both $f_0 : G_0 \to H_0$ and $f_1 : G_1 \to H_1$ are surjective.*

The proof is omitted. A similar statement is true for monomorphisms of graphs.

3.4.6 In **Set** an arrow that is both monic and epic is bijective (Theorems 3.3.3 and 3.4.2), and hence an isomorphism. In general, this need not happen. One example is the inclusion of **N** in **Z** in **Mon** described in 3.4.3 (an inverse would also have to be an inverse in **Set**, but there isn't one since the inclusion is not bijective). An easier example is the arrow from C to D in the category **2** in (2.1). It is both monic and epic (vacuously) but there is no arrow from D to C so it is not an isomorphism because there is no arrow in the category that could be its inverse.

3.4.7 An arrow $f : A \to B$ in a category is an isomorphism if it has an inverse $g : B \to A$ which must satisfy both the equations $g \circ f = \mathrm{id}_A$ and $f \circ g = \mathrm{id}_B$. If it only satisfies the second equation, $f \circ g = \mathrm{id}_B$, then f is a **left inverse** of g and (as you might expect) g is a **right inverse** of f.

3.4.8 Definition Suppose f has a right inverse g. Then f is called a **split epimorphism** (f is "split by g") and g is called a **split monomorphism**.

A split epimorphism is indeed an epimorphism: if $h \circ f = k \circ f$ and f has a right inverse g, then $h = h \circ f \circ g = k \circ f \circ g = k$, which is what is required for f to be an epimorphism. A dual proof shows that a split monomorphism is a monomorphism.

Using the usual axioms of set theory, every surjection in **Set** is a split epimorphism. For if $f : A \to B$, then choose, for each $b \in B$, some element $a \in A$ such that $f(a) = b$. The existence of such an a is guaranteed by surjectivity. Define $g(b)$ to be a. Then $f(g(b)) = f(a) = b$ for any $b \in B$, so $f \circ g = \mathrm{id}_B$.

The so-called axiom of choice is exactly what is required to make all those generally infinitely many choices. And in fact, one possible formulation of the axiom of choice is that every epimorphism split.

Epimorphisms in other categories may not be split. The function that includes the monoid of non-negative integers on addition in the monoid of all the integers on addition, which we mentioned in 3.4.3, certainly does not have a right inverse in the category of monoids, since it does not have a right inverse in the category of sets. There are plenty of examples of epimorphisms of monoids which *are* surjective which have no right inverse in the category of monoids, although of course they do in the category of sets. One such epimorphism of monoids is the function that takes the integers mod 4 on addition to the integers mod 2 on addition, with 0 and 2 going to 0 and 1 and 3 going to 1.

Unlike epis, which always split in the category of sets, monics in **Set** do not always split. Every arrow out of the empty set is monic and, save for the identity of $\emptyset$ to itself, is not split. On the other hand, every monic with nonempty source does split. We leave the details to you.

3.4.9 Hom sets The elementary categorical definitions given in the last section and this one can all be phrased in terms of hom set. In any category, $\mathrm{Hom}(A, B)$ is the set of arrows with source A and target B.

Thus a terminal object 1 satisfies the requirement that $\mathrm{Hom}(A, 1)$ is a singleton set for every object A, and an initial object 0 satisfies the dual requirement that $\mathrm{Hom}(0, A)$ is always a singleton. And $\mathrm{Hom}(1, A)$ is the set of constants (global elements) of A.

3.4.10 If $f : B \to C$, f induces a set function

$$\mathrm{Hom}(A, f) : \mathrm{Hom}(A, B) \to \mathrm{Hom}(A, C)$$

defined by composing by f on the left: for any $g \in \mathrm{Hom}(A, B)$, that is, for any $g : A \to B$, $\mathrm{Hom}(A, f)(g) = f \circ g$, which does indeed go from A to C.

Similarly, for any object D, $f : B \to C$ induces a set function

$$\mathrm{Hom}(f, D) : \mathrm{Hom}(C, D) \to \mathrm{Hom}(B, D)$$

(note the reversal) by requiring that $\mathrm{Hom}(f, D)(h) = h \circ f$ for $h \in \mathrm{Hom}(C, D)$.

In terms of these functions, we can state this proposition, which we leave to you to prove.

3.4.11 Proposition *An arrow $f : B \to C$ in a category*

(i) *is a monomorphism if and only if* $\mathrm{Hom}(A, f)$ *is injective for every object A;*

(ii) *is an epimorphism if and only if* $\mathrm{Hom}(f, D)$ *is injective (!) for every object D;*

(iii) *is a split monomorphism if and only if* $\mathrm{Hom}(f, D)$ *is surjective for every object D;*

(iv) *is a split epimorphism if and only if* $\mathrm{Hom}(A, f)$ *is surjective for every object A;*

(v) *is an isomorphism if and only if any one of the following equivalent conditions holds:*

  (a) *it is both a split epi and a mono;*

  (b) *it is both an epi and a split mono;*

  (c) $\mathrm{Hom}(A, f)$ *is bijective for every object A;*

(d) $\mathrm{Hom}(f, A)$ *is bijective for every object* A.

Although many categorical definitions can be given in terms of hom sets, no categorical definition *must* be; in fact, some mathematicians consider category theory to be a serious alternative to set theory as a foundation for mathematics (see many works of Lawvere, including [1963] and [1966] as well as [McLarty, 1989]) and for that purpose (which is not our purpose, of course), definition in terms of hom sets or any other sets must be avoided.

3.4.12 Discussion Categorical definitions, as illustrated in the simple ideas of the preceding sections, provide a method of abstract specification which has proved very useful in mathematics. They have, in particular, clarified concepts in many disparate branches of mathematics and provided as well a powerful unification of concepts across these branches.

The method of categorical definition is close in spirit to the modern attitude of computing science that programs and data types should be specified abstractly before being implemented and that the specification should be kept conceptually distinct from the implementation. We believe that the method of categorical definition is a type of abstract specification which is suitable for use in many areas of theoretical computing science. This is one of the major themes of this notes.

When a category $\mathscr{C}$ is a category of sets with structure, with the arrows being functions which preserve the structure, a categorical definition of a particular property does not involve the elements (in the standard sense of set theory) of the structure. Such definitions are said to be **element-free**, and that has been regarded as a great advantage of category theory.

Nevertheless, as we have seen, some definitions can be phrased in terms of *variable elements*. This allows us the option of using familiar modes of thinking about things in terms of elements even in general categories. In the case of the definition of monomorphism 3.3.2, the definition phrased in terms of variable elements is identical with the definition in **Set**. On the other hand, an epimorphism f (see 3.4.1) is a variable element with the property that any two different arrows out of its target must have different values at f. In some sense it is a variable element with a lot of variation. This is an example of a situation where the variable element point of view is not very familiar.

The idea of variable element has much in common with the way mathematicians and physicists once thought of variable quantities. Perhaps thirty years from now the variable element idea will be much more pervasive and the idea that an epimorphism is an element with a lot of variation will be the natural way to describe it.

4. Functors

A functor F from a category $\mathscr{C}$ to a category $\mathscr{D}$ is a graph homomorphism which preserves identities and composition. It plays the same role as monoid homomorphisms for monoids and monotone maps for posets: it preserves the structure that a category has. Functors have another significance, however: since one sort of thing a category can be is a mathematical workspace (see Preface), many of the most useful functors used by mathematicians are *transformations from one type of mathematics to another*. Less obvious, but perhaps more important, is the fact that many categories that are mathematically interesting appear as categories whose objects are a natural class of functors into the category of sets.

4.1 Functors

A functor is a structure-preserving map between categories, in the same way that a homomorphism is a structure-preserving map between graphs or monoids. Here is the formal definition.

4.1.1 Definition A **functor** $F : \mathscr{C} \to \mathscr{D}$ is a pair of functions $F_0 : \mathscr{C}_0 \to \mathscr{D}_0$ and $F_1 : \mathscr{C}_1 \to \mathscr{D}_1$ for which

F–1 If $f : A \to B$ in $\mathscr{C}$, then $F_1(f) : F_0(A) \to F_0(B)$ in $\mathscr{D}$.

F–2 For any object A of $\mathscr{C}$, $F_1(\mathrm{id}_A) = \mathrm{id}_{F_0(A)}$.

F–3 If $g \circ f$ is defined in $\mathscr{C}$, then $F_1(g) \circ F_1(f)$ is defined in $\mathscr{D}$ and $F_1(g \circ f) = F_1(g) \circ F_1(f)$.

By F–1, a functor is in particular a homomorphism of graphs. Following the practice for graph homomorphisms, the notation is customarily overloaded (see 1.2.2): if A is an object, $F(A) = F_0(A)$ is an object, and if f is an arrow, $F(f) = F_1(f)$ is an arrow. The notation for the constituents $F_0 : \mathscr{C}_0 \to \mathscr{D}_0$ and $F_1 : \mathscr{C}_1 \to \mathscr{D}_1$ is not standard, and we will use it only for emphasis.

4.1.2 Example It is easy to see that a monoid homomorphism $f : M \to N$ determines a functor from $C(M)$ to $C(N)$ as defined in 2.3.12. On objects, a homomorphism f must take the single object of $C(M)$ to the single object of $C(N)$, and F–1 is trivially verified since all arrows in $C(M)$ have the same domain and codomain and similarly for $C(N)$. Then F–2 and F–3 say precisely that f is a monoid homomorphism. Conversely, every functor is determined in this way by a monoid homomorphism.

4.1.3 Example Let us see what a functor from $C(S, \alpha)$ to $C(T, \beta)$ must be when (S, α) and (T, β) are posets as in 2.3.1. It is suggestive to write both relations α and β as '$\leq$' and the posets simply as S and T. Then there is exactly one arrow from x to y in S (or in T) if and only if $x \leq y$; otherwise there are no arrows from x to y.

Let $f : S \to T$ be the functor. F–1 says if there is an arrow from x to y, then there is an arrow from $f(x)$ to $f(y)$; in other words,

$$\text{if } x \leq y \text{ then } f(x) \leq f(y)$$

Thus f is a monotone map (see 2.4.2). F–2 and F–3 impose no additional conditions on f because they each assert the equality of two specified arrows between two specified objects and in a poset as category all arrows between two objects are equal.

4.1.4 The category of categories The category **Cat** has all small categories as objects and all functors between such categories as arrows. The composite of functors is their composite as graph homomorphisms: if $F : \mathscr{C} \to \mathscr{D}$ and $G : \mathscr{D} \to \mathscr{E}$, then $G \circ F : \mathscr{C} \to \mathscr{E}$ satisfies $G \circ F(C) = G(F(C))$ for any object C of $\mathscr{C}$, and $G \circ F(f) = G(F(f))$ for any arrow f of $\mathscr{C}$. Thus $(G \circ F)_i = G_i \circ F_i$ for $i = 0, 1$.

We note that the composition circle is usually omitted when composing functors so that we write $GF(C) = G(F(C))$.

It is sometimes convenient to refer to a category **CAT** which has all small categories and ordinary large categories as objects, and functors between them. Since trying to have **CAT** be an object of itself would raise delicate foundational questions, we do not attempt here a formal definition of **CAT**.

4.1.5 Properties of Cat We note some properties without proof.

The initial category has no objects and, therefore, no arrows. The terminal category has just one object and the identity arrow of that object. To any category $\mathscr{C}$ there is just one functor that takes every object to that single object and every arrow to that one arrow.

A functor is a monomorphism in **Cat** if and only if it is injective on both objects and arrows. The corresponding statement for epimorphisms is not true.

4.1.6 Example If $\mathscr{C}$ is a category, the functor

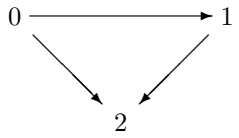
$$P_1 : \mathscr{C} \times \mathscr{C} \longrightarrow \mathscr{C}$$

(see 2.6.6) which takes an object (C, D) to C and an arrow $(f, g) : (C, D) \longrightarrow (C', D')$ to f is called the **first projection**. There is an analogous second projection functor P_2 taking an object or arrow to its second coordinate.

4.1.7 Example Let $\mathbf{2} + \mathbf{2}$ be the category that can be pictured as

$$0 \longrightarrow 1 \qquad 1' \longrightarrow 2$$

with no other nonidentity arrows, and the category $\mathbf{3}$ the one that looks like



(4.1)

Define the functor $F : \mathbf{2} + \mathbf{2} \longrightarrow \mathbf{3}$ to take 0 to 0, 1 and $1'$ to 1, and 2 to 2. Then what it does on arrows is forced.

Note that the image of F includes all of $\mathscr{D}$ except the composite arrow from $0 \longrightarrow 2$. This example shows that the image of a functor need not be a subcategory of the codomain category.

4.1.8 Example The inclusion map of a subcategory is a functor. As we pointed out in 3.3.14, the categorical point of view does not require that the object and arrows of a subcategory actually be objects and arrows of the bigger category, only that there be a monomorphism from the subcategory to the category. For example, **Set** is a subcategory of **Rel**: the monomorphic functor takes every set to itself and each function $f : S \longrightarrow T$ to its graph $\{(s, t) \mid t = f(s)\}$, which is indeed a relation from S to T.

This approach has the strange result that two different categories can each be regarded as subcategories of the other one.

4.1.9 Underlying functors Forgetting some of the structure in a category of structures and structure-preserving functions gives a functor called an **underlying functor** or **forgetful functor**. The functor $U : \mathbf{Mon} \longrightarrow \mathbf{Sem}$ which embeds the category of monoids into the category of semigroups by forgetting that a monoid has an identity is an example of an underlying functor.

Another example is the functor which forgets *all* the structure of a semigroup. This is a functor $U : \mathbf{Sem} \longrightarrow \mathbf{Set}$. There are lots of semigroups with the same set of elements; for example, the set $\{0, 1, 2\}$ is a semigroup on addition (mod 3) and also a different semigroup on multiplication (mod 3). The functor U applied to these two different semigroups gives the same set, so U is not injective on objects, in contrast to the forgetful functor from monoids to semigroups.

We will not give a formal definition of underlying functor. It is reasonable to expect any underlying functor U to be faithful (see 4.3.2 below) and that if f is an isomorphism and $U(f)$ is an identity arrow then f is an identity arrow.

4.1.10 Example A small graph has *two* underlying sets: its set of nodes and its set of arrows. Thus there is an underlying functor $U : \mathbf{Grf} \longrightarrow \mathbf{Set} \times \mathbf{Set}$ for which for a graph $\mathscr{G}$, $U(\mathscr{G}) = (G_0, G_1)$; an arrowset functor $A : \mathbf{Grf} \longrightarrow \mathbf{Set}$ which takes a graph to its set of arrows and a graph homomorphism to the corresponding function from arrows to arrows; and a similarly defined nodeset functor $N : \mathbf{Grf} \longrightarrow \mathbf{Set}$ which takes a graph to its set of nodes.

4.1.11 Example If you forget you can compose arrows in a category and you forget which arrows are the identities, then you have remembered only that the category is a graph. This gives an underlying functor $U : \mathbf{Cat} \to \mathbf{Grf}$, since every functor is a graph homomorphism although not vice versa.

As for graphs, there are also set-of-objects and set-of-arrows functors $O : \mathbf{Cat} \to \mathbf{Set}$ and $A : \mathbf{Cat} \to \mathbf{Set}$ which take a category to its set of objects and set of arrows respectively, and a functor to the appropriate set map.

4.1.12 Example In 2.6.10, we described the notion of a slice category $\mathscr{C}/A$ based on a category $\mathscr{C}$ and an object A. An object is an arrow $B \to A$ and an arrow from $f : B \to A$ to $g : C \to A$ is an arrow $h : B \to C$ for which

$$g \circ h = f$$

There is a functor $U : \mathscr{C}/A \to \mathscr{C}$ that takes the object $f : B \to A$ to B and the arrow h from $B \to A$ to $C \to A$ to $h : B \to C$. This is called the underlying functor of the slice. In the case that $\mathscr{C} = \mathbf{Set}$, an object $T \to S$ of $\mathbf{Set}/S$ for some set S is an S-indexed object, and the effect of the underlying functor is to forget the indexing.

4.1.13 Free functors The **free monoid functor** from **Set** to the category of monoids takes a set A to the free monoid $F(A)$, which is the Kleene closure A^* with concatenation as operation (see 2.3.9), and a function $f : A \to B$ to the function $F(f) = f^* : F(A) \to F(B)$ defined in 2.5.7.

To see that the free monoid functor is indeed a functor it is necessary to show that if $f : A \to B$ and $g : B \to C$, then $F(g \circ f) : F(A) \to F(C)$ is the same as $F(g) \circ F(f)$, which is immediate from the definition, and that it preserves identity arrows, which is also immediate.

The Kleene closure is itself a functor from **Set** to **Set**, taking A to A^* and f to f^*. It is the composite $U \circ F$ of the underlying functor $U : \mathbf{Mon} \to \mathbf{Set}$ and the free functor $F : \mathbf{Set} \to \mathbf{Mon}$, but of course it can be defined independently of U and F.

4.1.14 Example The free category on a graph is also the object part of a functor $F : \mathbf{Grf} \to \mathbf{Cat}$. What it does to a graph is described in 2.6.16. Suppose $\phi : \mathscr{G} \to \mathscr{H}$ is a graph homomorphism. The objects of the free category on a graph are the nodes of the graph, so it is reasonable to define $F(\phi)_0 = \phi_0$. Now suppose $(f_n, f_{n-1}, \ldots, f_1)$ is a path, that is, an arrow, in $F(\mathscr{G})$. Since functors preserve domain and codomain, we can define $F(\phi)_1(f_n, f_{n-1}, \ldots, f_1)$ to be $(\phi_1(f_n), \phi_1(f_{n-1}), \ldots, \phi_1(f_1))$ and know we get a path in $F(\mathscr{H})$. That F preserves composition of paths is also clear.

4.1.15 The map-lifting property The free category functor $F : \mathbf{Grf} \to \mathbf{Cat}$ and also other free functors, such as the free monoid functor (4.1.13), have a map lifting property called its **universal mapping property** which will be seen in Section 9.2 as the defining property of freeness. We will describe the property for free categories since we use it later. The free monoid case is done in detail in Proposition 9.1.2.

Let $\mathscr{G}$ be a graph and $F(\mathscr{G})$ the free category generated by $\mathscr{G}$. There is a graph homomorphism with the special name $\eta\mathscr{G} : \mathscr{G} \to U(F(\mathscr{G}))$ which includes a graph $\mathscr{G}$ into $U(F(\mathscr{G}))$, the underlying graph of the free category $F(\mathscr{G})$. The map $(\eta\mathscr{G})_0$ is the identity, since the objects of $F(\mathscr{G})$ are the nodes of $\mathscr{G}$. For an arrow f of $\mathscr{G}$, $(\eta\mathscr{G})_1(f)$ is the path (f) of length one. This is an inclusion arrow in the generalized categorical sense of 3.3.14, since f and (f) are really two distinct entities.

4.1.16 Proposition *Let $\mathscr{G}$ be a graph and $\mathscr{C}$ a category. Then for every graph homomorphism $h : \mathscr{G} \to U(\mathscr{C})$, there is a unique functor $\widehat{h} : F(\mathscr{G}) \to \mathscr{C}$ with the property that $U(\widehat{h}) \circ \eta\mathscr{G} = h$.*

Proof. If () is the empty path at an object a, we set $\widehat{h}() = \mathrm{id}_a$. For an object a of $F(\mathscr{G})$ (that is, node of $\mathscr{G}$), define $\widehat{h}(a) = h(a)$. And for a path $(a_n, a_{n-1}, \ldots, a_1)$, $\widehat{h}$ is 'map h':

$$\widehat{h}(a_n, a_{n-1}, \ldots, a_1) = (h(a_n), h(a_{n-1}), \ldots, h(a_1))$$

As noted in 2.1.1, there is a unique empty path for each node a of $\mathscr{G}$. Composing the empty path at a with any path p from a to b gives p again, and similarly on the other side.

4.1.17 Powerset functors Any set S has a powerset $\mathscr{P}S$, the set of all subsets of S. There are three different functors F for which F_0 takes a set to its powerset; they differ on what they do to arrows. One of them is fundamental in topos theory; that one we single out to be called the powerset functor.

If $f : A \to B$ is any set function and C is a subset of B, then the **inverse image** of C, denoted $f^{-1}(C)$, is the set of elements of A which f takes into C: $f^{-1}(C) = \{a \in A \mid f(a) \in C\}$. Thus f^{-1} is a function from $\mathscr{P}B$ to $\mathscr{P}A$.

Note that for a bijection f, the symbol f^{-1} is also used to denote the inverse function. Context makes it clear which is meant, since the input to the inverse image function must be a subset of the codomain of f, whereas the input to the actual inverse of a bijection must be an *element* of the codomain.

4.1.18 Definition The **powerset functor** $\mathscr{P} : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Set}$ takes a set S to the powerset $\mathscr{P}S$, and a set function $f : A \to B$ (that is, an arrow from B to A in $\mathbf{Set}^{\mathrm{op}}$) to the inverse image function $f^{-1} : \mathscr{P}B \to \mathscr{P}A$.

Although we will continue to use the notation f^{-1}, it is denoted f^* in much of the categorical literature.

To check that $\mathscr{P}$ is a functor requires showing that $\mathrm{id}_A^{-1} = \mathrm{id}_{\mathscr{P}A}$ and that if $g : B \to C$, then $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$, where both compositions take place in **Set**.

4.1.19 A functor $F : \mathscr{C}^{\mathrm{op}} \to \mathscr{D}$ is also called a **contravariant functor** from $\mathscr{C}$ to $\mathscr{D}$. As illustrated in the preceding definition, the functor is often defined in terms of arrows of $\mathscr{C}$ rather than of arrows of $\mathscr{C}^{\mathrm{op}}$. Opposite categories are most commonly used to provide a way of talking about contravariant functors as ordinary (**covariant**) functors: the opposite category in this situation is a purely formal construction of no independent interest (see 2.6.9).

4.1.20 The other two functors which take a set to its powerset are both covariant. The **direct** or **existential image** functor takes $f : A \to B$ to the function $f_* : \mathscr{P}A \to \mathscr{P}B$, where $f_*(A_0) = \{f(x) \mid x \in A_0\}$, the set of values of f on A_0. The **universal image** functor takes A_0 to those values of f which come *only* from A_0: formally, it takes $f : A \to B$ to $f_! : \mathscr{P}A \to \mathscr{P}B$, with

$$f_!(A_0) = \{y \in B \mid f(x) = y \text{ implies } x \in A_0\} = \{y \in B \mid f^{-1}(\{y\}) \subseteq A_0\}$$

4.1.21 Hom functors Let $\mathscr{C}$ be a category with an object C and an arrow $f : A \to B$. In 3.4.10, we defined the function $\mathrm{Hom}(C, f) : \mathrm{Hom}(C, A) \to \mathrm{Hom}(C, B)$ by setting $\mathrm{Hom}(C, f)(g) = f \circ g$ for every $g \in \mathrm{Hom}(C, A)$, that is for $g : C \to A$. We use this function to define the **covariant hom functor** $\mathrm{Hom}(C, -) : \mathscr{C} \to \mathbf{Set}$ as follows:

HF–1 $\mathrm{Hom}(C, -)(A) = \mathrm{Hom}(C, A)$ for each object A of $\mathscr{C}$;

HF–2 $\mathrm{Hom}(C, -)(f) = \mathrm{Hom}(C, f) : \mathrm{Hom}(C, A) \to \mathrm{Hom}(C, B)$ for $f : A \to B$.

The following calculations show that $\mathrm{Hom}(C, -)$ is a functor. For an object A, $\mathrm{Hom}(C, \mathrm{id}_A) : \mathrm{Hom}(C, A) \to \mathrm{Hom}(C, A)$ takes an arrow $f : C \to A$ to $\mathrm{id}_A \circ f = f$; hence $\mathrm{Hom}(C, \mathrm{id}_A) = \mathrm{id}_{\mathrm{Hom}(C,A)}$. Now suppose $f : A \to B$ and $g : B \to D$. Then for any arrow $k : C \to A$,

$$\begin{aligned}
\Big(\mathrm{Hom}(C, g) \circ \mathrm{Hom}(C, f)\Big)(k) &= \mathrm{Hom}(C, g)\Big(\mathrm{Hom}(C, f)(k)\Big) \\
&= \mathrm{Hom}(C, g)(f \circ k) \\
&= g \circ (f \circ k) \\
&= (g \circ f) \circ k \\
&= \mathrm{Hom}(C, g \circ f)(k)
\end{aligned}$$

In terms of variable elements, $\mathrm{Hom}(C, f)$ takes the variable elements of A with parameter set C to the variable elements of B with parameter set C.

There is a distinct covariant hom functor $\mathrm{Hom}(C, -)$ for each object C. In this expression, C is a parameter for a family of functors. The argument of each of these functors is indicated by the dash. An analogous definition in calculus would be to define the function which raises a real number to the nth power as $f(-) = (-)^n$ (here n is the parameter). One difference in the hom functor case is that the hom functor is overloaded and so has to be defined on two different kinds of things: objects and arrows.

4.1.22 Definition For a given object D, the **contravariant hom functor**

$$\mathrm{Hom}(-, D) : \mathscr{C}^{\mathrm{op}} \to \mathbf{Set}$$

is defined for each object A by

$$\mathrm{Hom}(-, D)(A) = \mathrm{Hom}(A, D)$$

and for each arrow $f : A \to B$,

$$\mathrm{Hom}(-, D)(f) = \mathrm{Hom}(f, D) : \mathrm{Hom}(B, D) \to \mathrm{Hom}(A, D)$$

Thus if $g : B \to D$, $\mathrm{Hom}(f, D)(g) = g \circ f$.

4.1.23 Definition The **two-variable hom functor**

$$\mathrm{Hom}(-, -) : \mathscr{C}^{\mathrm{op}} \times \mathscr{C} \to \mathbf{Set}$$

takes a pair (C, D) of objects of C to $\mathrm{Hom}(C, D)$, and a pair (f, g) of arrows with $f : C \to A$ and $g : B \to D$ to

$$\mathrm{Hom}(f, g) : \mathrm{Hom}(A, B) \to \mathrm{Hom}(C, D)$$

where for $h : A \to B$,

$$\mathrm{Hom}(f, g)(h) = g \circ h \circ f$$

which is indeed an arrow from C to D.

In this case we also use the product of categories as a formal construction to express functors of more than one argument. From the categorical point of view, a functor always has one argument, which as in the present case might well be an object in a product category (an ordered pair).

4.2 Actions

In this section, we discuss set-valued functors as a natural generalization of finite state machines. Set-valued functors also have theoretical importance in category theory because of the Yoneda Lemma (Section 5.7).

4.2.1 Monoid actions Let M be a monoid with identity 1 and let S be a set. An **action** of M on S is a function $\alpha : M \times S \to S$ for which

A–1 $\alpha(1, s) = s$ for all $s \in S$.

A–2 $\alpha(mn, s) = \alpha(m, \alpha(n, s))$ for all $m, n \in M$ and $s \in S$.

It is customary in mathematics to write ms for $\alpha(m, s)$; then the preceding requirements become

A′–1 $1s = s$ for all $s \in S$.

A′–2 $(mn)s = m(ns)$ for all $m, n \in M$ and $s \in S$.

When actions are written this way, S is also called an M**-set**. The same syntax ms for $m \in M$ and $s \in S$ is used even when different actions are involved. This notation is analogous to (and presumably suggested by) the notation $c\mathbf{v}$ for scalar multiplication, where c is a scalar and $\mathbf{v}$ is a vector.

It is useful to think of the set S as a **state space** and the elements of M as acting to induce **transitions** from one state to another.

4.2.2 Example One major type of action by a monoid is the case when the state space is a vector space and M is a collection of linear transformations closed under multiplication. However, in that case the linear structure (the fact that states can be added and multiplied by scalars) is extra structure which the definition above does not require. Our definition also does not require that there be any concept of continuity of transitions. Thus, the definition is very general and can be regarded as a *nonlinear, discrete* approach to state transition systems.

Less structure means, as always, that fewer theorems are true and fewer useful tools are available. On the other hand, less structure means that more situations fit the axioms, so that the theorems that are true and the tools that do exist work for more applications.

4.2.3 Definition Let M be a monoid with actions on sets S and T. An **equivariant map** from S to T is a function $\phi : S \to T$ with the property that $m\phi(s) = \phi(ms)$ for all $m \in M$ and $s \in S$. The identity function is an equivariant map and the composite of two equivariant maps is equivariant. This means that for each monoid M, monoid actions and equivariant maps form a category M–**Act**.

4.2.4 Actions as functors Let α be an action of a monoid M on a set S. Let $C(M)$ denote the category determined by M as in 2.3.12. The action α determines a functor $F_\alpha : C(M) \to \mathbf{Set}$ defined by:

AF–1 $F_\alpha(*) = S$.

AF–2 $F_\alpha(m) = s \mapsto \alpha(m, s)$ for $m \in M$ and $s \in S$.

This observation will allow us to generalize actions to categories in 4.2.6.

4.2.5 Finite state machines A particularly important example of a monoid action occurs in the study of finite state machines. Let A be a finite set, the **alphabet** of the machine, whose elements may be thought of as characters or tokens, and let S be another finite set whose elements are to be thought of as states. We assume there is a distinguished state $s_0 \in S$ called the **start state**, and a function $\phi : A \times S \to S$ defining a transition to a state for each token in A and each state in S. Such a system $\mathscr{M} = (A, S, s_0, \phi)$ is a **finite state machine**. Note that there is no question of imposing axioms such as A–1 and A–2 because A is not a monoid.

Any string w in A^* induces a sequence of transitions in the machine $\mathscr{M}$ starting at the state s_0 and ending in some state s. Precisely, we define a function $\phi^* : A^* \times S \to S$ by:

FA–1 $\phi^*((), s) = s$ for $s \in S$.

FA–2 $\phi^*((a)w, s) = \phi(a, \phi^*(w, s))$ for any $s \in S$, $w \in A^*$ and $a \in A$.

Recall that the free monoid $F(A)$ is the set A^* with concatenation as multiplication. The function ϕ^* as just defined is thus an action of $F(A)$ on S. The identity of A^* is the empty word () and by FA–1, $\phi^*((), a) = a$ for all $a \in A$, so A–1 follows. As for A–2, if we assume that

$$\phi^*(wv, m) = \phi^*(w, \phi^*(v, m))$$

for words w of length k, then

$$\begin{aligned}&\phi^*((a)wv, m) = \phi(a, \phi^*(wv, m))\\ &= \phi(a, \phi^*(w, \phi^*(v, m))) = \phi^*((a)w, \phi^*(v, m))\end{aligned}$$

The first and third equality are from the definition of ϕ, while the second is from the inductive hypothesis.

Finite state machines in the literature often have added structure. The state space may have a subset F of **acceptor states** (or **final states**). The subset L of A^* of strings which drive the machine from the start state to an acceptor state is then the set of strings, or language, which is **recognized** by the machine $\mathscr{M}$. This is the machine as **recognizer**. A compiler typically uses a finite state machine to recognize identifiers in the input file.

Another approach is to assume that the machine outputs a string of symbols (not necessarily in the same alphabet) for each state it enters or each transition it undergoes. This is the machine as **transducer**.

An elementary introduction to finite state machines may be found in [Lewis and Papadimitriou, 1981]. Two more advanced texts which use algebraic methods to study finite state machines (primarily as recognizers) are those by Eilenberg [1976] and Lallement [1979]. The latter book has many other applications of semigroup theory as well.

4.2.6 Set-valued functors as actions Suppose we wanted to extend the idea of an action by introducing typing. What would the result be?

To begin with, we would suppose that in addition to the state space S, there was a type set T and a function type : $S \to T$ that assigned to each element $s \in S$ an element type$(s) \in T$.

In describing the elements of M, one must say, for an $m \in M$ and $s \in S$, what is type(ms). Moreover, it seems that one might well want to restrict the types of the inputs on which a given m acts. In fact, although it might not be strictly necessary in every case, it seems clear that we can, without loss of

generality, suppose that each $m \in M$ acts on only one kind of input and produces only one kind of output. For if m acted on two types of output, we could replace m by two elements, one for each type. Thus we can imagine that there are two functions we will call input and output from M to T for which input(m) is the type of element that m acts on and output(m) is the type of $m(s)$ for an element s of type input(m).

In the untyped case, we had that M was a monoid, but here it is clearly appropriate to suppose that $m_1 * m_2$ is defined only when output(m_2) = input(m_1). It is reasonable to suppose that for each type t, there is an operation $1_t \in M$ whose input and output types are t and such that for any $m \in M$ of input type t, we have $m * 1_t = m$ and for any $m \in M$ of output type t, we have $1_t * m = m$.

As for the action, we will evidently wish to suppose that when $s \in S$ has type t and $m, m' \in M$ have input types t, t', respectively, and output types t', t'', respectively, then $m'(m(s)) = (m' * m)(s)$ and $1_t(s) = s$.

Now it will not have escaped the reader at this point that M and T together constitute a category $\mathscr{C}$ whose objects are the elements of T and arrows are the elements of M. The input and output functions are just the source and target arrows and the 1_t are the identities.

M and S make up exactly the data of a set-valued functor on $\mathscr{C}$. Define a functor $F : \mathscr{C} \to \mathbf{Set}$ by letting $F(t) = \{s \in S \mid \text{type}(s) = t\}$. If m is an arrow of $\mathscr{C}$, that is an element of M, let its input and output types be t and t', respectively. Then for F to be a functor, we require a function $F(m) : F(t) \to F(t')$. Naturally, we define $F(m)(s) = ms$, which indeed has type t'. The facts that F preserves composition and identities are an easy consequence of the properties listed above.

This construction can be reversed. Let $\mathscr{C}$ be a small category and suppose we have a functor $F : \mathscr{C} \to \mathbf{Set}$ for which $F(C)$ and $F(D)$ are disjoint whenever C and D are distinct objects of $\mathscr{C}$ (this disjointness requirement is necessary to have a category, but can be forced by a simple modification of F – see Section 5.7). Then we can let T be the set of objects of $\mathscr{C}$, M the set of arrows and $S = \bigcup_{t \in T} F(t)$. The rest of the definitions are evident and we leave them to the reader.

Thus if $\mathscr{C}$ is a small category, a functor $F : \mathscr{C} \to \mathbf{Set}$ is an action which generalizes the concept of monoid acting on a set.

4.2.7 Example For any given object C of a category $\mathscr{C}$, the hom functor Hom($C, -$) (see 4.1.21) is a particular example of a set-valued functor. When the category $\mathscr{C}$ is a monoid, it is the action by left multiplication familiar in semigroup theory. A theorem generalizing the Cayley theorem for groups is true, too (see 5.7.5).

4.2.8 Variable sets It may be useful to think of a set-valued functor $F : \mathscr{C} \to \mathbf{Set}$ as an action, not on a typed set, but on a single *variable* set. The objects of $\mathscr{C}$ form a parameter space for the variation of the set being acted upon. Another way of saying this is that each object of $\mathscr{C}$ is a *point of view*, that the set being acted upon looks different from different points of view, and the arrows of $\mathscr{C}$ are changes in point of view (as well as inducing transitions). See [Barr, McLarty and Wells, 1985].

4.3 Types of functors

Since **Cat** is a category, we already know about some types of functors. Thus a functor $F : \mathscr{C} \to \mathscr{D}$ is an isomorphism if there is a functor $G : \mathscr{D} \to \mathscr{C}$ which is inverse to F. This implies that F is bijective on objects and arrows and conversely a functor which is bijective on objects and arrows is an isomorphism.

We have already pointed out that a functor is a monomorphism in **Cat** if and only if it is injective on both objects and arrows. Epimorphisms in **Cat** need not be surjective, since the example in 3.4.3 is actually an epimorphism in **Cat** between the categories determined by the monoids.

4.3.1 Full and faithful We will now consider properties of functors which are more intrinsic to **Cat** than the examples just given.

Any functor $F : \mathscr{C} \to \mathscr{D}$ induces a set mapping

$$\text{Hom}_{\mathscr{C}}(A, B) \to \text{Hom}_{\mathscr{D}}(F(A), F(B))$$

for each pair of objects A and B of $\mathscr{C}$. This mapping takes an arrow $f : A \to B$ to $F(f) : F(A) \to F(B)$.

4.3.2 Definition A functor $F : \mathscr{C} \to \mathscr{D}$ is **faithful** if the induced mapping is injective on every hom set.

Thus if $f : A \to B$ and $g : A \to B$ are different arrows, then $F(f) \neq F(g)$. However, it is allowed that $f : A \to B$ and $g : C \to D$ may be different arrows, with $F(A) = F(C)$, $F(B) = F(D)$ and $F(f) = F(g)$, provided that either $A \neq C$ or $B \neq D$.

4.3.3 Example Underlying functors are typically faithful. Two different monoid homomorphisms between the same two monoids must be different as set functions.

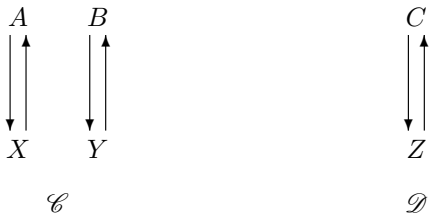
On the other hand, consider the set $\{0, 1, 2\}$. It has two different monoid structures via addition and multiplication (mod 3) (and many other monoid structures, too), but the two corresponding identity homomorphisms are the same as set functions (have the same underlying function). Thus underlying functors need not be injective.

4.3.4 Definition A functor $F : \mathscr{C} \to \mathscr{D}$ is **full** if the induced mapping is surjective for every hom set.

A full functor need not be surjective on either objects or arrows. A full subcategory (2.6.3) is exactly one whose embedding is a full and faithful functor.

That the underlying functor from the category of semigroups to the category of sets is not full says exactly that not every set function between semigroups is a semigroup homomorphism. Note that this functor is surjective on objects, since every set can be made into a semigroup by letting $xy = x$ for every pair x and y of elements.

4.3.5 Example The functor $F : \mathscr{C} \to \mathscr{D}$ which takes A and B to C and X and Y to Z (and so is forced on arrows) in the picture below (which omits identity arrows) is *not* full. That is because $\mathrm{Hom}(A, B)$ is empty, but $\mathrm{Hom}(F(A), F(B)) = \mathrm{Hom}(C, C)$ has an arrow in it – the identity arrow. This functor is faithful even though not injective, since two arrows between the same two objects do not get identified.



(4.2)

4.3.6 Preservation of properties A functor $F : \mathscr{C} \to \mathscr{D}$ **preserves** a property P of arrows if whenever f has property P, so does $F(f)$.

4.3.7 Examples The fact that a monomorphism in the category of monoids must be injective can be worded as saying that the underlying functor preserves monomorphisms (since an injective function in **Set** is a monomorphism). The statement that an epimorphism in **Mon** need not be surjective is the same as saying that the underlying functor does not preserve epimorphisms.

As another example, consider the functor $F : \mathbf{2} \to \mathbf{Set}$ ($\mathbf{2}$ is shown in (2.1), page 6) defined by $C \mapsto \{1, 2\}$, $D \mapsto \{3, 4\}$ and the arrow from C to D going to the constant function $1 \mapsto 3$, $2 \mapsto 3$ from $F(C)$ to $F(D)$. The arrow from C to D is monic and epic (vacuously) but its value in **Set** takes 1 and 2 both to 3, so is not injective and hence not a monomorphism. It is also not an epimorphism. Thus F preserves neither monomorphisms nor epimorphisms.

The story is different for isomorphisms. (Note that the arrow from C to D in $\mathbf{2}$ is not an isomorphism!)

4.3.8 Proposition *Every functor preserves isomorphisms.*

Proof. This is because the concept of isomorphism is defined in terms of equations involving composition and identity. If $f : A \to B$ is an isomorphism with inverse g, then $F(g)$ is the inverse of $F(f)$. One of the two calculations necessary to prove this is that $F(g) \circ F(f) = F(g \circ f) = F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$; the other calculation is analogous. □

In fact, an analogous proof shows that every functor preserves split monos and split epis.

4.3.9 Definition A functor $F : \mathscr{C} \to \mathscr{D}$ **reflects** a property P of arrows if whenever $F(f)$ has property P then so does f (for *any* arrow that F takes to $F(f)$).

It follows from 2.5.5 and the definition of isomorphism (3.1.4) that a bijective semigroup homomorphism must be an isomorphism. That is the same as saying that the underlying functor from **Sem** to **Set** reflects isomorphisms. The same remark applies to **Mon**. The underlying functor from the category of posets and monotone maps does not reflect isomorphisms.

A full and faithful functor reflects isomorphisms, but in fact it does a bit more than that, as described by the following proposition.

4.3.10 Proposition *Let $F : \mathscr{C} \to \mathscr{D}$ be full and faithful, and suppose A and B are objects of $\mathscr{C}$ and $u : F(A) \to F(B)$ is an isomorphism in $\mathscr{D}$. Then there is a unique isomorphism $f : A \to B$ for which $F(f) = u$.*

Proof. By fullness, there are arrows $f : A \to B$ and $g : B \to A$ for which $F(f) = u$ and $F(g) = u^{-1}$. Then

$$F(g \circ f) = F(g) \circ F(f) = u^{-1} \circ u = \mathrm{id}_{F(A)} = F(\mathrm{id}_A)$$

But F is faithful, so $g \circ f = \mathrm{id}_A$. A similar argument shows that $f \circ g = \mathrm{id}_B$, so that g is the inverse of f. □

4.3.11 Corollary *A full and faithful functor reflects isomorphisms.*

4.3.12 Corollary *Let $F : \mathscr{C} \to \mathscr{D}$ be a full and faithful functor. If $F(A) = F(B)$ for objects A and B of $\mathscr{C}$, then A and B are isomorphic.*

Proof. Apply Proposition 4.3.10 to the identity arrow from $F(A)$ to $F(A) = F(B)$. □

4.3.13 You can also talk about a functor preserving or reflecting a property of objects. For example, since a terminal object in **Mon** is a one-element monoid and a one-element set is a terminal object, the underlying functor from **Mon** to **Set** preserves terminal objects. It also reflects terminal objects. It does not preserve initial objects, but it does reflect initial objects although vacuously: the empty set is the only initial object in **Set** and the underlying set of a monoid cannot be empty since it must have an identity element. We leave the details to you.

4.4 Equivalences

In this section we define what it means for two categories to be equivalent. The correct concept turns out to be weaker than requiring that they be isomorphic – that is, that there is a functor from one to the other which has an inverse in **Cat**. In order to understand the issues involved, we first take a close look at the construction of the category corresponding to a monoid in Section 2.3.12. It turns out to be a functor.

4.4.1 Monoids and one-object categories For each monoid M we constructed a small category $C(M)$ in 2.3.12. We make the choice mentioned there that the one object of $C(M)$ is M. Note that although an element of $C(M)$ is now an arrow from M to M, it is *not* a set function.

For each monoid homomorphism $h : M \to N$, construct a functor $C(h) : C(M) \to C(N)$ as follows:

CF–1 On objects, $C(h)(M) = N$.

CF–2 $C(h)$ must be exactly the same as h on arrows (elements of M).

It is straightforward to see that $C(h)$ is a functor and that this construction makes C a functor from **Mon** to the full subcategory of **Cat** of categories with exactly one object. We will denote this full subcategory as **Ooc**.

There is also a functor $U : \mathbf{Ooc} \to \mathbf{Mon}$ going the other way.

UO–1 For a category $\mathscr{C}$ with one object, $U(\mathscr{C})$ is the monoid whose elements are the arrows of $\mathscr{C}$ and whose binary operation is the composition of $\mathscr{C}$.

UO–2 If $F : \mathscr{C} \to \mathscr{D}$ is a functor between one-object categories, $U(F) = F_1$, that is, the functor F on arrows.

The functors U and C are not inverse to each other, and it is worthwhile to see in detail why.

The construction of C is in part arbitrary. We needed to regard each monoid as a category with one object. The choice of the elements of M to be the arrows of the category is obvious, but what should be the one object? We chose M itself, but we could have chosen some other thing, such as the set $\{e\}$, where e is the identity of M. The only real requirement is that it not be an element of M (such as its identity) in order to avoid set-theoretic problems caused by the category being an element of itself. The consequence is that we have given a functor $C : \mathbf{Mon} \to \mathbf{Ooc}$ in a way which required arbitrary choices.

The arbitrary choice of one object for $C(M)$ means that if we begin with a one-object category $\mathscr{C}$, construct $M = U(\mathscr{C})$, and then construct $C(M)$, the result will not be the same as C unless it happens that the one object of $\mathscr{C}$ is M. Thus $C \circ U \neq \mathrm{id}_{\mathbf{Ooc}}$, so that U is not the inverse of C. (In this case $U \circ C$ is indeed $\mathrm{id}_{\mathbf{Mon}}$.)

C is not surjective on objects, since not every small category with one object is in the image of C; in fact a category $\mathscr{D}$ is $C(M)$ for some monoid M only if the single object of $\mathscr{D}$ is actually a monoid and the arrows of $\mathscr{D}$ are actually the arrows of that monoid. This is entirely contrary to the spirit of category theory: we are talking about specific elements rather than specifying behavior. Indeed, in terms of specifying behavior, the category of monoids and the category of small categories with one object ought to be essentially the same thing.

The fact that C is not an isomorphism of categories is a signal that isomorphism is the wrong idea for capturing the concept that two categories are essentially the same. However, every small category with one object is *isomorphic* to one of those constructed as $C(M)$ for some monoid M. This is the starting point for the definition of equivalence.

4.4.2 Definition A functor $F : \mathscr{C} \to \mathscr{D}$ is an **equivalence of categories** if there are:

E–1 A functor $G : \mathscr{D} \to \mathscr{C}$.

E–2 A family $u_C : C \to G(F(C))$ of isomorphisms of $\mathscr{C}$ indexed by the objects of $\mathscr{C}$ with the property that for every arrow $f : C \to C'$ of $\mathscr{C}$, $G(F(f)) = u_{C'} \circ f \circ u_C^{-1}$.

E–3 A family $v_D : D \to F(G(D))$ of isomorphisms of $\mathscr{D}$ indexed by the objects of $\mathscr{D}$, with the property that for every arrow $g : D \to D'$ of $\mathscr{D}$, $F(G(g)) = v_{D'} \circ g \circ v_D^{-1}$.

If F is an equivalence of categories, the functor G of E–1 is called a **pseudo-inverse** of F. That the functor C of 4.4.1 is an equivalence (with pseudo-inverse U) is left as an exercise.

The idea behind the definition is that not only is every object of $\mathscr{D}$ isomorphic to an object in the image of F, but the isomorphisms are compatible with the arrows of $\mathscr{D}$; and similarly for $\mathscr{C}$.

4.4.3 Example An isomorphism of categories is an equivalence of categories, and its inverse is its pseudo-inverse.

4.4.4 Example Let $\mathscr{C}$ be the category with two objects A and B, their identities, and two other arrows $i : A \to B$ and $j : B \to A$ that are inverse isomorphisms between the objects:

$$A \underset{j}{\overset{i}{\rightleftarrows}} B \tag{4.3}$$

Let $\mathscr{D} = \mathbf{1}$ be the category with one object E and its identity arrow. Then $\mathscr{C}$ and $\mathscr{D}$ are equivalent. The unique functor from $\mathscr{C}$ to $\mathscr{D}$ has two pseudo-inverses, each taking the unique object of $\mathscr{D}$ to one of the two isomorphic objects of $\mathscr{C}$.

We give the details for one of these. Let $F : \mathscr{C} \to \mathscr{D}$ be the functor that takes A and B to E and $G : \mathscr{D} \to \mathscr{C}$ the functor that takes E to A. The family required by E–2 consists of $u_A = \mathrm{id}_A$ and $u_B = j$. That required by E–3 consists of id_E. We have for example

$$G(F(i)) = G(\mathrm{id}_E) = \mathrm{id}_A = j \circ i \circ \mathrm{id}_A = u_B \circ i \circ {u_A}^{-1}$$

The other equations required by E–2 and E–3 are similar or easier.

4.4.5 Theorem *Let $F : \mathscr{C} \to \mathscr{D}$ be an equivalence of categories and $G : \mathscr{D} \to \mathscr{C}$ a pseudo-inverse to F. Then F and G are full and faithful.*

Proof. Actually, something more is true: if F and G are functors for which E–2 is true, then F is faithful. For suppose $f, f' : C \to C'$ in $\mathscr{C}$ and $F(f) = F(f')$ in $\mathscr{D}$. Then $G(F(f)) = G(F(f'))$ in $\mathscr{C}$, so that

$$f = u_{C'}^{-1} \circ G(F(f)) \circ u_C = u_{C'}^{-1} \circ G(F(f')) \circ u_C = f'$$

Thus F is faithful. A symmetric argument shows that if E–3 is true then G is faithful.

Now suppose $F : \mathscr{C} \to \mathscr{D}$ is an equivalence of categories and $G : \mathscr{D} \to \mathscr{C}$ is a pseudo-inverse to F. We now know that F and G are faithful. To show that F is full, suppose that $g : F(C) \to F(C')$ in $\mathscr{D}$. We must find $f : C \to C'$ in $\mathscr{C}$ for which $F(f) = g$. Let $f = u_{C'}^{-1} \circ G(g) \circ u_C$. Then a calculation using E–2 shows that $G(F(f)) = G(g)$. Since G is faithful, $F(f) = g$. □

Proposition 4.3.10 implies that an equivalence of categories does not take nonisomorphic objects to isomorphic ones.

An alternative definition of equivalence sometimes given in the literature uses the concept of representative functor. A functor $F : \mathscr{C} \to \mathscr{D}$ is **representative** if every object of $\mathscr{D}$ is isomorphic to an object in the image of F. (Thus a subcategory is representative in the sense of Definition 3.1.9 if the inclusion functor is representative.) Then a functor $F : \mathscr{C} \to \mathscr{D}$ is an equivalence if it is full, faithful, and representative. This definition can be proved equivalent to ours. The proof requires the axiom of choice.

4.4.6 Inequivalence For any property P of a category that can be defined in terms of composition and identities, if $\mathscr{C}$ and $\mathscr{D}$ are equivalent categories, then either they both have property P or neither of them does. This is an imprecise statement; in particular, a property preserved by equivalence can require that two arrows be the same but it cannot require that two objects be the same. A formal language that expresses the properties preserved by equivalence is given by Freyd and Scedrov [1990], sections 1.39–1.3(10). See also [Bergman and Berman, 1998]. Reference added

This observation provides a way to show that two categories are not equivalent. For example, **Set** and **Mon** are not equivalent because there is no arrow in **Set** from the terminal object to the initial object, but in **Mon** the initial and terminal objects are isomorphic. Similarly **Set** and the category of posets and monotone functions are not equivalent because there are only two nonisomorphic sets that have only one automorphism (the empty set and a singleton set), but there are many nonisomorphic posets that have only one automorphism, for example any two totally ordered finite posets of different cardinality.

5. Diagrams and naturality

Commutative diagrams are the categorist's way of expressing equations. Natural transformations are maps between functors; one way to think of them is as a deformation of one construction (construed as a functor) into another.

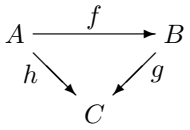
5.1 Diagrams

We begin with diagrams in a graph and discuss commutativity later.

5.1.1 Definition Let $\mathscr{I}$ and $\mathscr{G}$ be graphs. A **diagram** in $\mathscr{G}$ of **shape** $\mathscr{I}$ is a homomorphism $D : \mathscr{I} \longrightarrow \mathscr{G}$ of graphs. $\mathscr{I}$ is called the **shape graph** of the diagram D.

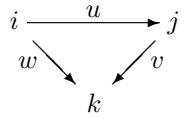
We have thus given a new name to a concept which was already defined (not uncommon in mathematics). A diagram is a graph homomorphism from a different point of view.

5.1.2 Example At first glance, Definition 5.1.1 may seem to have little to do with what are informally called diagrams, for example



(5.1)

The connection is this: a diagram in the sense of Definition 5.1.1 is pictured on the page with a drawing of nodes and arrows as for example in Diagram (5.1), which could be the picture of a diagram D with shape graph



(5.2)

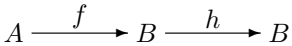
defined by $D(i) = A$, $D(j) = B$, $D(k) = C$, $D(u) = f$, $D(v) = g$ and $D(w) = h$.

5.1.3 Example Here is an example illustrating some subtleties involving the concept of diagram. Let $\mathscr{G}$ be a graph with objects A, B and C (and maybe others) and arrows $f : A \longrightarrow B$, $g : B \longrightarrow C$ and $h : B \longrightarrow B$. Consider these two diagrams, where here we use the word 'diagram' informally:

(a) (b)

(5.3)

These are clearly of different shapes (again using the word 'shape' informally). But the diagram



(5.4)

is the same shape as (5.3)(a) even though as a graph it is the same as (5.3)(b).

To capture the difference thus illustrated between a graph and a diagram, we introduce two shape graphs

$\mathscr{I}$ $\mathscr{J}$

(5.5)

(where, as will be customary, we use numbers for the nodes of shape graphs). Now diagram (5.3)(a) is

seen to be the diagram $D : \mathscr{I} \to \mathscr{G}$ with $D(1) = A$, $D(2) = B$, $D(3) = C$, $D(u) = f$ and $D(v) = g$; whereas diagram (5.3)(b) is $E : \mathscr{J} \to \mathscr{G}$ with $E(1) = A$, $E(2) = B$, $E(u) = f$ and $E(w) = h$. Moreover, Diagram (5.4) is just like D (has the same shape), except that v goes to h and 3 goes to B.

5.1.4 Our definition in 5.1.1 of a diagram as a graph homomorphism, with the domain graph being the shape, captures both the following ideas:

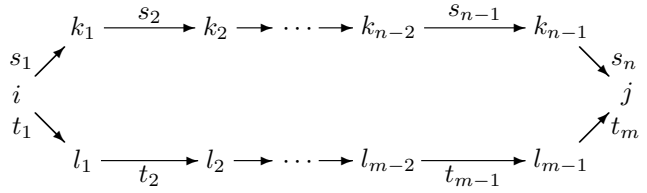
(i) A diagram can have repeated labels on its nodes and (although the examples did not show it) on its arrows, and

(ii) Two diagrams can have the same labels on their nodes and arrows but be of different shapes: Diagrams (5.3)(b) and (5.4) are *different diagrams* because they have different shapes.

5.1.5 Commutative diagrams When the target graph of a diagram is the underlying graph of a category some new possibilities arise, in particular the concept of commutative diagram, which is the categorist's way of expressing equations.

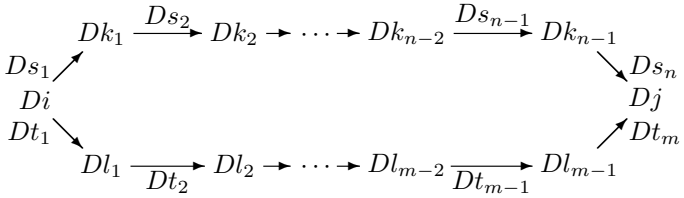
In this situation, we will not distinguish in notation between the category and its underlying graph: if $\mathscr{I}$ is a graph and $\mathscr{C}$ is a category we will refer to a diagram $D : \mathscr{I} \to \mathscr{C}$.

We say that D is **commutative** (or **commutes**) provided for any nodes i and j of $\mathscr{I}$ and any two paths



(5.6)

from i to j in $\mathscr{I}$, the two paths



(5.7)

compose to the same arrow in $\mathscr{C}$. This means that

$$Ds_n \circ Ds_{n-1} \circ \ldots \circ Ds_1 = Dt_m \circ Dt_{m-1} \circ \ldots \circ Dt_1$$

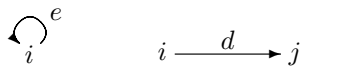
5.1.6 Much ado about nothing There is one subtlety to the definition of commutative diagram: what happens if one of the numbers m or n in Diagram (5.7) should happen to be 0? If, say, $m = 0$, then we interpret the above equation to be meaningful only if the nodes i and j are the same (you go nowhere on an empty path) and the meaning in this case is that

$$Ds_n \circ Ds_{n-1} \circ \ldots \circ Ds_1 = \mathrm{id}_{D_i}$$

(you do nothing on an empty path). In particular, a diagram D based on the graph

commutes if and only if $D(e)$ is the identity arrow from $D(i)$ to $D(i)$.

Note, and note well, that both shape graphs



(a) (b)

have models that one might think to represent by the diagram

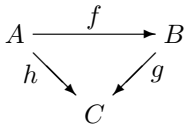
but the diagram based on (a) commutes if and only if $f = \mathrm{id}_A$, while the diagram based on (b) commutes automatically (no two nodes have more than one path between them so the commutativity condition is vacuous).

We will always picture diagrams so that distinct nodes of the shape graph are represented by distinct (but possibly identically labeled) nodes in the picture. Thus a diagram based on (b) in which d goes to f and i and j both go to A will be pictured as

$$A \xrightarrow{f} A$$

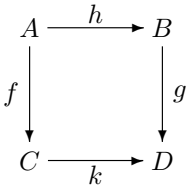
In consequence, one can always deduce the shape graph of a diagram from the way it is pictured, except of course for the actual names of the nodes and arrows of the shape graph.

5.1.7 Examples of commutative diagrams – and others The prototypical commutative diagram is the triangle



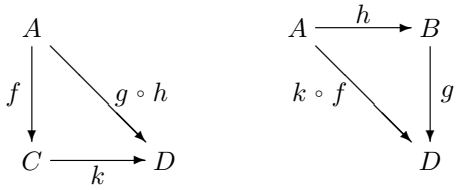
(5.8)

that commutes if and only if h is the composite $g \circ f$. The reason this is prototypical is that any commutative diagram – unless it involves an empty path – can be replaced by a set of commutative triangles. This fact is easy to show and not particularly enlightening, so we are content to give an example. The diagram



(5.9)

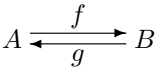
commutes if and only if the two diagrams



(5.10)

commute (in fact if and only if either one does).

5.1.8 Example An arrow $f : A \longrightarrow B$ is an isomorphism with inverse $g : B \longrightarrow A$ if and only if



(5.11)

commutes. The reason for this is that for this diagram to commute, the two paths () and (g, f) from A to A must compose to the same value in the diagram, which means that $g \circ f = \mathrm{id}_A$. A similar observation shows that $f \circ g$ must be id_B.

5.1.9 Graph homomorphisms by commutative diagrams The definition of graph homomorphism in 1.2.1 can be expressed by a commutative diagram. Let $\phi = (\phi_0, \phi_1)$ be a graph homomorphism from $\mathscr{G}$ to $\mathscr{H}$. For any arrow $u : m \longrightarrow n$ in $\mathscr{G}$, 1.2.1 requires that $\phi_1(u) : \phi_0(m) \longrightarrow \phi_0(n)$ in $\mathscr{H}$. This says that $\phi_0(\text{source}(u)) = \text{source}(\phi_1(u))$, and a similar statement about targets. In other words, these diagrams must commute:

$$\begin{array}{ccc} G_1 & \xrightarrow{\phi_1} & H_1 \\ {\scriptstyle\text{source}}\downarrow & & \downarrow{\scriptstyle\text{source}} \\ G_0 & \xrightarrow[\phi_0]{} & H_0 \end{array} \qquad \begin{array}{ccc} G_1 & \xrightarrow{\phi_1} & H_1 \\ {\scriptstyle\text{target}}\downarrow & & \downarrow{\scriptstyle\text{target}} \\ G_0 & \xrightarrow[\phi_0]{} & H_0 \end{array} \tag{5.12}$$

In these two diagrams the two arrows labeled 'source' are of course different functions; one is the source function for $\mathscr{G}$ and the other for $\mathscr{H}$. A similar remark is true of 'target'.

5.1.10 This point of view provides a pictorial proof that the composite of two graph homomorphisms is a graph homomorphism (see 2.4.1). If $\phi : \mathscr{G} \longrightarrow \mathscr{H}$ and $\psi : \mathscr{H} \longrightarrow \mathscr{K}$ are graph homomorphisms, then to see that $\psi \circ \phi$ is a graph homomorphism requires checking that the outside rectangle below commutes, and similarly with target in place of source:

$$\begin{array}{ccccc} G_1 & \xrightarrow{\phi_1} & H_1 & \xrightarrow{\psi_1} & K_1 \\ {\scriptstyle\text{source}}\downarrow & & \downarrow{\scriptstyle\text{source}} & & \downarrow{\scriptstyle\text{source}} \\ G_0 & \xrightarrow[\phi_0]{} & H_0 & \xrightarrow[\psi_0]{} & K_0 \end{array} \tag{5.13}$$

The outside rectangle commutes because the two squares commute. This can be checked by tracing (mentally or with a finger or pointer) the paths from G_1 to K_0 to verify that

$$\text{source} \circ \psi_1 \circ \phi_1 = \psi_0 \circ \text{source} \circ \phi_1 \tag{5.14}$$

because the right square commutes, and that

$$\psi_0 \circ \text{source} \circ \phi_1 = \psi_0 \circ \phi_0 \circ \text{source} \tag{5.15}$$

because the left square commutes. The verification process just described is called 'chasing the diagram'. Of course, one can verify the required fact by writing the equations (5.14) and (5.15) down, but those equations hide the source and target information given in Diagram (5.13) and thus provide a possibility of writing an impossible composite down. For many people, Diagram (5.13) is much easier to remember than equations (5.14) and (5.15). However, diagrams are more than informal aids; they are formally-defined mathematical objects just like automata and categories.

The proof in 2.6.10 that the composition of arrows in a slice gives another arrow in the category can be represented by a similar diagram:

$$\begin{array}{ccccc} C & \xrightarrow{h} & C' & \xrightarrow{h'} & C'' \\ & {\scriptstyle f}\searrow & \downarrow{\scriptstyle f'} & \swarrow{\scriptstyle f''} & \\ & & A & & \end{array}$$

These examples are instances of **pasting** commutative diagrams together to get bigger ones.

5.1.11 Associativity by commutative diagrams The fact that the multiplication in a monoid or semigroup is associative can be expressed as the assertion that a certain diagram in **Set** commutes.

Let S be a semigroup. Define the following functions:

(i) $\text{mult} : S \times S \longrightarrow S$ satisfies $\text{mult}(x, y) = xy$.

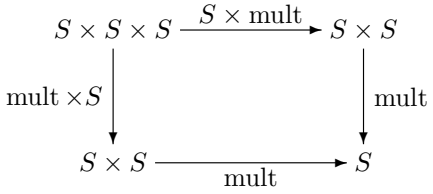
(ii) $S \times \text{mult} : S \times S \times S \to S \times S$ satisfies

$$(S \times \text{mult})(x, y, z) = (x, yz)$$

(iii) $\text{mult} \times S : S \times S \times S \to S \times S$ satisfies

$$(\text{mult} \times S)(x, y, z) = (xy, z)$$

That the following diagram commutes is exactly the associative law.

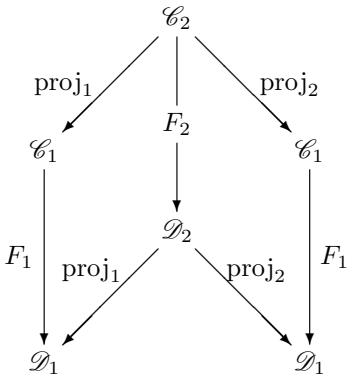


(5.16)

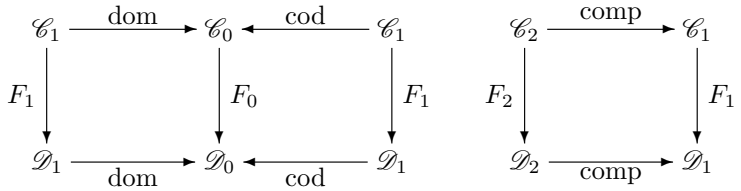
5.1.12 Normally, associativity is expressed by the equation $x(yz) = (xy)z$ for all x, y, z in the semigroup. The commutative diagram expresses this same fact *without the use of variables*. Of course, we did use variables in defining the functions involved, but we remedy that deficiency in Chapter 6 when we give a categorical definition of products.

Another advantage of using diagrams to express equations is that diagrams show the source and target of the functions involved. This is not particularly compelling here but in other situations the two-dimensional picture of the compositions involved makes it much easier to follow the discussion.

5.1.13 Functors by commutative diagrams We express the definition of functor using commutative diagrams. Let $\mathscr{C}$ and $\mathscr{D}$ be categories with sets of objects $\mathscr{C}_0$ and $\mathscr{D}_0$, sets of arrows $\mathscr{C}_1$ and $\mathscr{D}_1$, and sets of composable pairs of arrows $\mathscr{C}_2$ and $\mathscr{D}_2$, respectively. A functor $F : \mathscr{C} \to \mathscr{D}$ consists of functions $F_0 : \mathscr{C}_0 \to \mathscr{D}_0$, $F_1 : \mathscr{C}_1 \to \mathscr{D}_1$ along with the uniquely determined function $F_2 : \mathscr{C}_2 \to \mathscr{D}_2$ such that



commutes. In addition, the following diagrams must commute:



5.1.14 Diagrams as functors In much of the categorical literature, a diagram in a category $\mathscr{C}$ is a functor $D : \mathscr{E} \to \mathscr{C}$ where $\mathscr{E}$ is a category. Because of Proposition 4.1.16, a graph homomorphism into a category extends uniquely to a functor based on the free category generated by the graph, so that diagrams in our sense generate diagrams in the functorial sense. On the other hand, any functor is a graph homomorphism on the underlying graph of its domain (although not conversely!), so that every diagram in the sense of functor is a diagram in the sense of graph homomorphism.

5.2 Natural transformations

5.2.1 Unary operations In Section 5.1 we saw that diagrams in a category are graph homomorphisms to the category from a different point of view. Now we introduce a third way to look at graph homomorphisms to a category, namely as models. To give an example, we need a definition.

5.2.2 Definition A **unary operation** on a set S is a function $u : S \to S$.

This definition is by analogy with the concept of binary operation on a set. A set with a unary operation is a (very simple) algebraic structure, which we call a **u-structure**. If the set is S and the operation is $f : S \to S$, we say that (S, f) is a u-structure, meaning (S, f) denotes a u-structure whose underlying set is S, and whose unary operation is f. This uses positional notation in much the same way as procedures in many computer languages: the first entry in the expression '(S, f)' is the name of the underlying set of the u-structure and the second entry is the name of its operation.

5.2.3 A homomorphism of u-structures should be a function which preserves the structure. There is really only one definition that is reasonable for this idea: if (S, u) and (T, v) are u-structures, $f : S \to T$ is a **homomorphism of u-structures** if $f(u(s)) = v(f(s))$ for all $s \in S$. Thus this diagram must commute:

$$\begin{array}{ccc} S & \xrightarrow{f} & T \\ {\scriptstyle u}\downarrow & & \downarrow{\scriptstyle v} \\ S & \xrightarrow[f]{} & T \end{array} \tag{5.17}$$

It is not difficult to show that the composite of two homomorphisms of u-structures is another one, and that the identity map is a homomorphism, so that u-structures and homomorphisms form a category.

5.2.4 Models of graphs We now use the concept of u-structure to motivate the third way of looking at graph homomorphisms to a category.

5.2.5 Let $\mathscr{U}$ be the graph with one node u_0 and one arrow e:

Let us define a graph homomorphism $D : \mathscr{U} \to \mathbf{Set}$ as follows: $D(u_0) = \mathbf{R}$ and $D(e) = x \mapsto x^2$. Now $(\mathbf{R}, x \mapsto x^2)$ is a u-structure, and the notation we have introduced in 5.2.1 tells us that we have chosen $\mathbf{R}$ to be its underlying set and $x \mapsto x^2$ to be its unary operation. Except for the arbitrary names 'u_0' and 'e', the graph homomorphism D communicates the same information: '$\mathbf{R}$ is the particular set chosen to be the value of u_0, and $x \mapsto x^2$ is the particular function chosen to be the value of e.'

In this sense, a u-structure is essentially the same thing as a diagram in **Set** of shape $\mathscr{U}$: a u-structure 'models' the graph $\mathscr{U}$. This suggests the following definition.

5.2.6 Definition A **model** M of a graph $\mathscr{G}$ is a graph homomorphism $M : \mathscr{G} \to \mathbf{Set}$.

A monoid can be defined as a model involving a graph homomorphism (and other ingredients) using the concept of finite product (see [Barr and Wells, 1999], Chapter 7). We had to introduce u-structures here to have an example for which we had the requisite techniques.

Both category theory and mathematical logic have concepts of 'model'. Both are formalisms attempting to make precise the relationship between the (formal) description of a mathematical structure and the structure itself. In logic, the precise description (the syntax) is given by a logical theory; in category theory by sketches or by categories regarded as theories. Good introductions to various aspects of categorical logic and model theory are given by [Makkai and Reyes, 1977], [Lambek and Scott, 1986], [Makkai and Paré, 1990] and [Adámek and Rosičky, 1994].

5.2.7 Example As another example, consider this graph:

$$\mathsf{a} \underset{\mathsf{target}}{\overset{\mathsf{source}}{\rightrightarrows}} \mathsf{n} \tag{5.18}$$

A model M of this graph consists of sets $G_0 = M(\mathsf{n})$ and $G_1 = M(\mathsf{a})$ together with functions source $= M(\mathsf{source}) : G_1 \to G_0$ and target $= M(\mathsf{target}) : G_1 \to G_0$. To understand what this structure is, imagine a picture in which there is a dot corresponding to each element of G_0 and an arrow corresponding to each element $a \in G_1$ which goes from the dot corresponding to source(a) to the one corresponding to target(a). It should be clear that the picture so described is a graph and thus the graph (5.18) is a graph whose models are graphs!

5.2.8 Models in arbitrary categories The concept of model can be generalized to arbitrary categories: if $\mathscr{C}$ is any category, a **model of $\mathscr{G}$ in $\mathscr{C}$** is a graph homomorphism from $\mathscr{G}$ to $\mathscr{C}$. For example, a model of the graph for u-structures in the category of posets and monotone maps is a poset and a monotone map from the poset to itself.

In these notes, the bare word 'model' always means a model in **Set**.

5.2.9 Natural transformations between models of a graph In a category, there is a natural notion of an arrow from one model of a graph to another. This usually turns out to coincide with the standard definition of homomorphism for that kind of structure.

5.2.10 Definition Let $D, E : \mathscr{G} \to \mathscr{C}$ be two models of the same graph in a category. A **natural transformation** $\alpha : D \to E$ is given by a family of arrows αa of $\mathscr{C}$ indexed by the nodes of $\mathscr{G}$ such that:

NT–1 $\alpha a : Da \to Ea$ for each node a of $\mathscr{G}$.

NT–2 For any arrow $s : a \to b$ in $\mathscr{G}$, the diagram

$$\begin{array}{ccc} Da & \xrightarrow{\alpha a} & Ea \\ {\scriptstyle Ds}\downarrow & & \downarrow{\scriptstyle Es} \\ Db & \xrightarrow[\alpha b]{} & Eb \end{array} \tag{5.19}$$

commutes.

The commutativity of the diagram in NT–2 is referred to as the **naturality condition** on α. The arrow αa for an object a is the **component** of the natural transformation α at a.

Note that you talk about a natural transformation from D to E only if D and E have the same domain (here $\mathscr{G}$) as well as the same codomain (here $\mathscr{C}$) and if, moreover, the codomain is a category. In this situation, it is often convenient to write $\alpha : D \to E : \mathscr{G} \to \mathscr{C}$.

5.2.11 Definition Let D, E and F be models of $\mathscr{G}$ in $\mathscr{C}$, and $\alpha : D \to E$ and $\beta : E \to F$ natural transformations. The **composite** $\beta \circ \alpha : D \to F$ is defined componentwise: $(\beta \circ \alpha)a = \beta a \circ \alpha a$.

5.2.12 Proposition *The composite of two natural transformations is also a natural transformation.*

Proof. The diagram that has to be shown commutative is the outer rectangle of

$$\begin{array}{ccccc} Da & \xrightarrow{\alpha a} & Ea & \xrightarrow{\beta a} & Fa \\ {\scriptstyle Ds}\downarrow & & \downarrow{\scriptstyle Es} & & \downarrow{\scriptstyle Fs} \\ Db & \xrightarrow[\alpha b]{} & Eb & \xrightarrow[\beta b]{} & Fb \end{array} \tag{5.20}$$

for each arrow $s : a \to b$ in $\mathscr{G}$. The rectangle commutes because the two squares do; the squares commute as a consequence of the naturality of α and β. □

It is interesting that categorists began using modes of reasoning like that in the preceding proof because objects of categories generally lacked elements; now one appreciates them for their own sake *because* they allow element-free (and thus variable-free) arguments.

5.2.13 It is even easier to show that there is an identity natural transformation between any model D and itself, defined by $(\mathrm{id}_D)a = \mathrm{id}_{Da}$. We then have the following proposition, whose proof is straightforward.

5.2.14 Proposition *The models of a given graph $\mathscr{G}$ in a given category $\mathscr{C}$, and the natural transformations between them, form a category. We denote this category by* $\mathbf{Mod}(\mathscr{G}, \mathscr{C})$.

5.2.15 Example The natural transformations between models in **Set** of the u-structure graph $\mathscr{U}$ defined in 5.2.5 are exactly the homomorphisms of u-structures defined in 5.2.3. The graph described in 5.2.5 has one object u_0 and one arrow e, so that a natural transformation from a model D to a model E has only one component which is a function from $D(u_0)$ to $E(u_0)$. If we set $S = D(u_0)$, $u = D(e)$, $T = E(u_0)$, $v = E(e)$, and we define $\alpha u_0 = f$, this is the single component of a natural transformation from D to E. Condition NT–2 in 5.2.10 coincides in this case with the diagram in 5.2.3: the naturality condition is the same as the definition of homomorphism of u-structures. It follows that the category of u-structures and homomorphisms is essentially $\mathbf{Mod}(\mathscr{U}, \mathbf{Set})$.

5.2.16 Example A homomorphism of graphs is a natural transformation between models of the graph

$$\mathsf{a} \underset{\text{target}}{\overset{\text{source}}{\rightrightarrows}} \mathsf{n}$$

The two graphs in Diagram (5.12) are the two necessary instances (one for the source and the other for the target) of Diagram (5.19). In a similar way, Diagram (5.20), used to show that the composite of two natural transformations is a natural transformation, reduces in this case to the commutativity of Diagram (5.13): specifically, the only possibilities (other than those in which s is an identity arrow) for a and b in Diagram (5.20) are $a = \mathsf{a}$ and $b = \mathsf{n}$, giving two diagrams shaped like Diagram (5.20), one for $s =$ source (that is Diagram (5.13)) and the other for $s =$ target.

5.2.17 Example A model of the graph

$$0 \xrightarrow{\ u\ } 1 \tag{5.21}$$

in an arbitrary category $\mathscr{C}$ is essentially the same as an arrow in $\mathscr{C}$ (see 5.2.23 below). A natural transformation from the model represented by the arrow $f : A \to B$ to the one represented by $g : C \to D$ is a pair of arrows $h : A \to C$ and $k : B \to D$ making a commutative diagram:

$$\begin{array}{ccc} A & \xrightarrow{\ h\ } & C \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[\ k\ ]{} & D \end{array} \tag{5.22}$$

The component at 0 is h and the component at 1 is k. The category of models in $\mathscr{C}$ is called the **arrow category** of $\mathscr{C}$; it is often denoted $\mathscr{C}^{\to}$.

5.2.18 Example Let $\mathscr{G}$ be the graph with two nodes and no arrows, and $\mathscr{C}$ any category. Then $\mathbf{Mod}(\mathscr{G}, \mathscr{C})$ is isomorphic to $\mathscr{C} \times \mathscr{C}$.

5.2.19 Natural isomorphisms A natural transformation $\alpha : F \to G : \mathscr{G} \to \mathscr{D}$ is called a **natural isomorphism** if there is a natural transformation $\beta : G \to F$ which is an inverse to α in the category $\mathbf{Mod}(\mathscr{G}, \mathscr{D})$. Natural isomorphisms are often called **natural equivalences**.

5.2.20 Example The arrow $(h,k) : f \longrightarrow g$ in the arrow category of a category $\mathscr{C}$, as shown in (5.22), is a natural isomorphism if and only if h and k are both isomorphisms in $\mathscr{C}$. This is a special case of an important fact about natural isomorphisms, which we now state.

5.2.21 Theorem *Suppose $F : \mathscr{G} \longrightarrow \mathscr{D}$ and $G : \mathscr{G} \longrightarrow \mathscr{D}$ are models of $\mathscr{G}$ in $\mathscr{D}$ and $\alpha : F \longrightarrow G$ is a natural transformation of models. Then α is a natural isomorphism if and only if for each node a of $\mathscr{G}$, $\alpha a : F(a) \longrightarrow G(a)$ is an isomorphism of $\mathscr{D}$.*

Proof. Suppose α has an inverse $\beta : G \longrightarrow F$ in $\mathbf{Mod}(\mathscr{G}, \mathscr{D})$. Then for any node a, by Definition 5.2.11, Definition 5.2.13, and the definition of inverse,

$$\alpha a \circ \beta a = (\alpha \circ \beta)a = \mathrm{id}_G\, a = \mathrm{id}_{G(a)}$$

and

$$\beta a \circ \alpha a = (\beta \circ \alpha)a = \mathrm{id}_F\, a = \mathrm{id}_{F(a)}$$

which means that the arrow βa is the inverse of the arrow αa, so that αa is an isomorphism in $\mathscr{D}$ as required.

Conversely, suppose that for each node a of $\mathscr{G}$, $\alpha a : F(a) \longrightarrow G(a)$ is an isomorphism of $\mathscr{D}$. The component of the inverse β at a node a is defined by letting $\beta a = (\alpha a)^{-1}$. This is the only possible definition, but it must be shown to be natural. Let $f : a \longrightarrow b$ be an arrow of the domain of F and G. Then we have

$$\begin{aligned} Ff \circ (\alpha a)^{-1} &= (\alpha b)^{-1} \circ (\alpha b) \circ Ff \circ (\alpha a)^{-1} \\ &= (\alpha b)^{-1} \circ Gf \circ (\alpha a) \circ (\alpha a)^{-1} \\ &= (\alpha b)^{-1} \circ Gf \end{aligned}$$

which says that β is natural. The second equality uses the naturality of α. □

5.2.22 Monic natural transformations Let $\alpha : F \longrightarrow G$ be a natural transformation between models of $\mathscr{G}$ in $\mathscr{D}$. Suppose each component of α is a monomorphism in $\mathscr{D}$. Then it is easy to prove that α is a monomorphism in $\mathbf{Mod}(\mathscr{G}, \mathscr{D})$. The converse is not true (see Section 4.2.21 of [Barr and Wells, 1999]).

5.2.23 'Essentially the same' In 5.2.17, we said that a model in an arbitrary category $\mathscr{C}$ of the graph (5.21) is 'essentially the same' as an arrow in $\mathscr{C}$. This is common terminology and usually refers implicitly to an equivalence of categories. We spell it out in this case.

Let us say that for a category $\mathscr{C}$, $\mathscr{C}'$ is the category whose objects are the arrows of $\mathscr{C}$ and for which an arrow from f to g is a pair (h,k) making Diagram (5.22) commute.

A model M of the graph (5.21) in a category $\mathscr{C}$ specifies the objects $M(0)$ and $M(1)$ and the arrow $M(u)$. $M(u)$ has domain $M(0)$ and codomain $M(1)$. But the domain and codomain of an arrow in a category are uniquely determined by the arrow. So that the only necessary information is which arrow $M(u)$ is.

Now we can define a functor $F : \mathscr{C}^{\longrightarrow} \longrightarrow \mathscr{C}'$. On objects it take M to $M(u)$. The remarks in the preceding paragraph show that this map on objects is bijective. If $M(u) = f$ and $N(u) = g$, an arrow from M to N in $\mathscr{C}^{\longrightarrow}$ and an arrow from f to g in $\mathscr{C}'$ are the same thing – a pair (h,k) making Diagram (5.22) commute. So we say F is the identity on arrows. It is straightforward to see that F is actually an isomorphism of categories.

In most texts, the arrow category $\mathscr{C}^{\longrightarrow}$ is defined the way we defined $\mathscr{C}'$.

When being careful, one would say as above that a model in $\mathscr{C}$ of the graph (5.21) is essentially the same as an arrow in $\mathscr{C}$, and that a u-structure is essentially the same as a model of $\mathscr{U}$ (as in 5.2.15). Frequently, one says more bluntly that a model of (5.21) in $\mathscr{C}$ *is* an arrow in $\mathscr{C}$ and that a u-structure *is* a model of $\mathscr{U}$ (and 'is an $\mathbf{N}$-set'). This usage is perhaps based on the conception that the description 'model of (5.21) in $\mathscr{C}$' and 'arrow in $\mathscr{C}$' are two ways of describing the same mathematical object, which exists independently of any particular description. Not all mathematicians share this conception of mathematical objects.

5.3 Natural transformations between functors

A functor is among other things a graph homomorphism, so a natural transformation between two functors is a natural transformation of the corresponding graph homomorphisms. The following proposition is an immediate consequence of 5.2.12.

5.3.1 Proposition *If $\mathscr{C}$ and $\mathscr{D}$ are categories, the functors from $\mathscr{C}$ to $\mathscr{D}$ form a category with natural transformations as arrows.*

We denote this category by $\mathbf{Func}(\mathscr{C}, \mathscr{D})$. Other common notations for it are $\mathscr{D}^{\mathscr{C}}$ and $[\mathscr{C}, \mathscr{D}]$. Tennent [1986] provides an exposition of the use of functor categories for programming language semantics.

Of course, the *graph* homomorphisms from $\mathscr{C}$ to $\mathscr{D}$, which do not necessarily preserve the composition of arrows in $\mathscr{C}$, also form a category $\mathbf{Mod}(\mathscr{C}, \mathscr{D})$ (see 5.2.14), of which $\mathbf{Func}(\mathscr{C}, \mathscr{D})$ is a full subcategory.

A natural transformation from one functor to another is a special case of a natural transformation from one graph homomorphism to another, so the ideas we have presented concerning natural transformations between graph homomorphisms apply to natural transformations between functors as well. In particular, Theorems 5.2.12 and 5.2.21 are true of natural transformations of functors.

If $\mathscr{C}$ is not a small category (see 2.1.5), then $\mathbf{Func}(\mathscr{C}, \mathscr{D})$ may not be locally small (see 2.1.7). This is a rather esoteric question that will not concern us in these notes since we will have no occasion to form functor categories of that sort.

We motivated the concept of natural transformation by considering models of graphs, and most of the discussion in the rest of this section concerns that point of view. Historically, the concept first arose for functors and not from the point of view of models.

5.3.2 Examples We have already described some examples of natural transformations, as summed up in the following propositions.

In 4.1.15, we defined the graph homomorphism $\eta\mathscr{G} : \mathscr{G} \longrightarrow U(F(\mathscr{G}))$ which includes a graph $\mathscr{G}$ into $U(F(\mathscr{G}))$, the underlying graph of the free category $F(\mathscr{G})$.

5.3.3 Proposition *The family of arrows $\eta\mathscr{G}$ form a natural transformation from the identity functor on* **Grf** *to $U \circ F$, where U is the underlying graph functor from* **Cat** *to* **Grf**.

The proof is left as an exercise.

In 4.4.2, we defined the concept of equivalence of categories.

5.3.4 Proposition *A functor $F : \mathscr{C} \longrightarrow \mathscr{D}$ is an equivalence of categories with pseudo-inverse $G : \mathscr{D} \longrightarrow \mathscr{C}$ if and only if $G \circ F$ is naturally isomorphic to $\mathrm{id}_{\mathscr{C}}$ and $F \circ G$ is naturally isomorphic to $\mathrm{id}_{\mathscr{D}}$.*

Proof. Conditions E–2 and E–3 of 4.4.2 can be recast as the statement that $G(F(f)) \circ u_C = u_{C'} \circ f$ and that $F(G(g)) \circ v_D = v_{D'} \circ g$, in other words that the following diagrams commute:

$$\begin{array}{ccc} C & \xrightarrow{u_C} & G(F(C)) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle G(F(f))} \\ C' & \xrightarrow[u_{C'}]{} & G(F(C')) \end{array} \qquad \begin{array}{ccc} D & \xrightarrow{v_D} & F(G(D)) \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle F(G(g))} \\ D' & \xrightarrow[v_{D'}]{} & F(G(D')) \end{array}$$

In this form, they are the statements that u is a natural transformation from $\mathrm{id}_{\mathscr{C}}$ to $G \circ F$ and that v is a natural transformation from $\mathrm{id}_{\mathscr{D}} \longrightarrow F \circ G$. Since each component of u and each component of v is an isomorphism, u and v are natural equivalences. □

5.3.5 Example Let $\alpha : M \times S \longrightarrow S$ and $\beta : M \times T \longrightarrow T$ be two actions by a monoid M (see 4.2.1). Let $\phi : S \longrightarrow T$ be an equivariant map. If F and G are the functors corresponding to α and β, as defined in 4.2.4, then ϕ is the (only) component of a natural transformation from F to G. Conversely, the only component of any natural transformation from F to G is an equivariant map between the corresponding actions.

5.3.6 Example Let $U : \mathbf{Mon} \to \mathbf{Set}$ be the underlying functor from the category of monoids. Define $U \times U : \mathbf{Mon} \to \mathbf{Set}$ as follows:

(i) For a monoid M, $(U \times U)(M) = U(M) \times U(M)$.

(ii) For a monoid homomorphism $h : M \to N$,

$$(U \times U)(h)(m, n) = (h(m), h(n))$$

Then monoid multiplication is a natural transformation from $U \times U$ to U. Formally: Let $\mu : U \times U \to U$ be the family of maps whose value at a monoid M is the function $\mu M : U(M) \times U(M) \to U(M)$ defined by $\mu M(m, m') = mm'$, the product of m and m' in M. Then μ is a natural transformation. (The function μM is *not* in general a monoid homomorphism, unless M is commutative.)

It is instructive to see why μ is a natural transformation. Let $h : M \to N$ be a monoid homomorphism. We must show that the following diagram commutes:

$$\begin{array}{ccc} (U \times U)(M) & \xrightarrow{\mu M} & U(M) \\ {\scriptstyle (U \times U)(h)}\downarrow & & \downarrow{\scriptstyle h} \\ (U \times U)(N) & \xrightarrow[\mu N]{} & U(N) \end{array} \tag{5.23}$$

The top route takes an element $(m, m') \in (U \times U)(M)$ to $h(mm')$. The lower route takes it to $h(m)h(m')$. The commutativity of the diagram then follows from the fact that h is a homomorphism.

5.3.7 Example Let $\mathscr{C}$ be a category. A **subfunctor** of a functor $F : \mathscr{C} \to \mathbf{Set}$ is a functor $G : \mathscr{C} \to \mathbf{Set}$ with the property that for each object C of $\mathscr{C}$, $G(C) \subseteq F(C)$ and such that for each arrow $f : C \to C'$ and each element $x \in GC$, we have that $Gf(x) = Ff(x)$. It is straightforward to check that the inclusion function $i_C : G(C) \to F(C)$ is a natural transformation.

5.3.8 Example Let B be a fixed set. We define a functor $R : \mathbf{Set}^{\,\mathrm{op}} \to \mathbf{Set}$ such that for a set A, $R(A) = \mathrm{Rel}(A, B)$, the set of relations from A to B. (A relation from A to B is essentially set of ordered pairs in $A \times B$.) For a set function $F : A' \to A$ and relation $\alpha \in \mathrm{Rel}(A, B)$, define $R(F)(\alpha)$ to be the relation $\alpha' \in \mathrm{Rel}(A', B)$ defined by $a'\alpha' b$ if and only if $F(a')\alpha b$. It is easy to see that this makes $R : \mathbf{Set}^{\,\mathrm{op}} \to \mathbf{Set}$ a functor. (Note that $R(A) = \mathbf{Rel}(A, B)$, but R is *not* $\mathrm{Hom}_{\mathbf{Rel}}(-, B)$.)

For each A, let $\phi_A : \mathrm{Rel}(A, B) \to \mathrm{Hom}(A, \mathscr{P}B)$ be the bijection that takes a relation α from A to B to the function that takes an element $a \in A$ to the set $\{b \in B \mid a\alpha b\}$. (You should check that this is a bijection.) If we check that the functions ϕ_A are the components of a natural transformation from R to $\mathrm{Hom}(A, \mathscr{P}B)$, the transformation will automatically be a natural isomorphism by Theorem 5.2.21. To show that it is natural, let $\alpha \in \mathbf{Rel}(A, B)$ and $a' \in A'$. Then

$$\begin{array}{rcccl} \mathrm{Hom}(F, \mathscr{P}B)\,(\phi_A(\alpha)(a')) & = & \phi_A(\alpha)\,(F(a')) & = & \{b \mid F(a')\alpha b\} \\ & = & \{b \mid a'\alpha' b\} & = & \phi_{A'}\,(R(F)(a')) \end{array}$$

as required.

This natural isomorphism can be taken to be the defining property of a topos (Section 11.2.2).

5.3.9 Example For each set S, let $\{\}S : S \to \mathscr{P}S$ be the function which takes an element x of S to the singleton subset $\{x\}$. Then $\{\}$ is a natural transformation from the identity functor on **Set** to the direct image powerset functor $\mathscr{P}$. (See 4.1.17.)

However, $\{\}$ is not a natural transformation from the identity functor on **Set** to the universal image powerset functor (see 4.1.20), and it does not even make sense to ask whether it is a natural transformation to the inverse image powerset functor.

5.4 Natural transformations involving lists

Many of the operations on lists available in functional programming languages can be seen as natural transformations involving the Kleene closure or list functor (4.1.13). x

5.4.1 Example One can apply the Kleene closure twice to get the list of lists functor that takes a set A to A^{**}. An element of A^{**} is a list of lists. For example, if $A = \{a, b\}$, one of the elements of A^{**} is $w = ((a, b), (b, b, a), (), (a))$. If $f : A \longrightarrow B$ is a function, f^{**} takes w to $((f(a), f(b)), (f(b), f(b), f(a)), (), (f(a)))$.

The operation of **flattening** a list simply concatenates the lists in the list; for example, $\text{flatten}(w) = (a, b, b, b, a, a)$. Of course, flatten is a distinct function for each set A; if we write $\text{flatten}_A : A^{**} \longrightarrow A^*$ for each set A, then flatten is a natural transformation from ** to *, as you can see by checking that this diagram commutes for each function $f : A \longrightarrow B$:

$$\begin{array}{ccc} A^{**} & \xrightarrow{\text{flatten}_A} & A^* \\ {\scriptstyle f^{**}}\downarrow & & \downarrow{\scriptstyle f^*} \\ B^{**} & \xrightarrow[\text{flatten}_B]{} & B^* \end{array} \tag{5.24}$$

5.4.2 Example Another operation in functional programming languages consists of applying a binary operation to a list. This is called **reduce**, **apply** or **fold**. We shall consider only the case when the binary operation is associative. (When it is not associative, some choice is made about how to associate the list.) This gives a natural transformation from $F \circ U$ to $\text{id}_{\mathbf{Mon}}$, where $F : \mathbf{Set} \longrightarrow \mathbf{Mon}$ is the free monoid functor and $U : \mathbf{Mon} \longrightarrow \mathbf{Set}$ is the underlying set functor. For example, if M is a monoid with elements k, m and n, then $\text{reduce}(k, k, n, m) = k^2nm$, the product of the list using the operation of M. In this case, naturality means that for any monoid homomorphism $h : M \longrightarrow N$, $h \circ \text{reduce}_M = \text{reduce}_N \circ F(U(h))$, which is easily checked.

Note that although both reduce and flatten take lists as arguments, reduce_M is a monoid homomorphism with domain $F(U(M))$ (whose elements are lists), whereas flatten_A is a set function with domain A^{**}. This is reflected by the fact that, when implemented in a programming language, reduce takes an operation as well as a list as argument, but flatten takes only a list.

These and other list operations can often be generalized to sets of expressions instead of sets of lists. In particular, flatten in this more general sense is one of the fundamental constituents of triples (Section 10.1), namely μ.

More about these ideas may be found in [Bird, 1986] and [Spivey, 1989].

5.5 Natural transformations of graphs

We now consider some natural transformations involving the category **Grf** of graphs and homomorphisms of graphs.

5.5.1 Example The map which takes an arrow of a graph to its source is a natural transformation from A to N. (See 4.1.10.) The same is true for targets. Actually, every operation in any multisorted algebraic structure gives a natural transformation. Example 5.3.6 was another example of this. See [Linton, 1969b], [Linton, 1969a].

5.5.2 Example In 4.1.10, we defined the functor $N : \mathbf{Grf} \longrightarrow \mathbf{Set}$. It takes a graph $\mathscr{G}$ to its set G_0 of nodes and a homomorphism ϕ to ϕ_0. Now pick a graph with one node $*$ and no arrows and call it $\mathscr{E}$. Let $V = \text{Hom}_{\mathbf{Grf}}(\mathscr{E}, -)$.

A graph homomorphism from the graph $\mathscr{E}$ to an arbitrary graph $\mathscr{G}$ is evidently determined by the image of $\mathscr{E}$ and that can be any node of $\mathscr{G}$. In other words, nodes of $\mathscr{G}$ are 'essentially the same thing' as graph homomorphisms from $\mathscr{E}$ to $\mathscr{G}$, that is, as the elements of the set $V(\mathscr{G})$.

We can define a natural transformation $\alpha : V \longrightarrow N$ by defining

$$\alpha\mathscr{G}(f) = f_0(*)$$

where $\mathscr{G}$ is a graph and $f : \mathscr{E} \to \mathscr{G}$ is a graph homomorphism (arrow of **Grf**). There must be a naturality diagram (5.19) for each arrow of the source category, which in this case is **Grf**. Thus to see that α is natural, we require that for each graph homomorphism $g : \mathscr{G}_1 \to \mathscr{G}_2$, the diagram

$$\begin{array}{ccc} V\mathscr{G}_1 & \xrightarrow{Vg} & V\mathscr{G}_2 \\ {\scriptstyle \alpha\mathscr{G}_1}\downarrow & & \downarrow{\scriptstyle \alpha\mathscr{G}_2} \\ N\mathscr{G}_1 & \xrightarrow[Ng]{} & N\mathscr{G}_2 \end{array}$$

commutes. Now Ng is g_0 (the node map of g) by definition, and the value of V (which is a hom functor) at a homomorphism g composes g with a graph homomorphism from the graph $\mathscr{E}$. Then we have, for a homomorphism $f : \mathscr{E} \to \mathscr{G}_1$ (i.e., an element of the upper left corner of the diagram),

$$(\alpha\mathscr{G}_2 \circ Vg)(f) = \alpha\mathscr{G}_2(g \circ f) = (g \circ f)_0(*)$$

while

$$(Ng \circ \alpha\mathscr{G}_1)(f) = Ng(f_0(*)) = g_0(f_0(*))$$

and these are equal from the definition of composition of graph homomorphisms.

The natural transformation α is in fact a natural isomorphism. This shows that N is naturally isomorphic to a hom functor. Such functors are called 'representable', and are considered in greater detail in 5.7.1.

5.5.3 Connected components A node a can be **connected** to the node b of a graph $\mathscr{G}$ if it is possible to get from a to b following a sequence of arrows of $\mathscr{G}$ in either direction. In order to state this more precisely, let us say that an arrow a 'has' a node n if n is the domain or the codomain (or both) of a. Then a is connected to b means that there is a sequence $(c_0, c_1, \ldots, c_n)$ of arrows of $\mathscr{G}$ with the property that a is a node (either the source or the target) of c_0, b is a node of c_n, and for $i = 1, \ldots, n$, c_{i-1} and c_i have a node in common. We call such a sequence an **undirected path** between a and b.

It is a good exercise to see that 'being connected to' is an equivalence relation. (For reflexivity: a node is connected to itself by the empty sequence.) An equivalence class of nodes with respect to this relation is called a **connected component** of the graph $\mathscr{G}$, and the set of connected components is called $W\mathscr{G}$.

Connected components can be defined for categories in the same way as for graphs. In that case, each connected component is a full subcategory.

If $f : \mathscr{G} \to \mathscr{H}$ is a graph homomorphism and if two nodes a and b are in the same component of $\mathscr{G}$, then $f(a)$ and $f(b)$ are in the same component of $\mathscr{H}$; this is because f takes an undirected path between a and b to an undirected path between $f(a)$ and $f(b)$. Thus the arrow f induces a function $Wf : W\mathscr{G} \to W\mathscr{H}$, namely the one which takes the component of a to the component of $f(a)$; and this makes W a functor from **Grf** to **Set**.

For a graph $\mathscr{G}$, let $\beta\mathscr{G} : N\mathscr{G} \to W\mathscr{G}$ be the set function which takes a node of $\mathscr{G}$ to the component of $\mathscr{G}$ that contains that node. (The component is the *value* of $\beta\mathscr{G}$ at the node, not the codomain.) Then $\beta : N \to W$ is a natural transformation. It is instructive to check the commutativity of the requisite diagram.

5.6 Combining natural transformations and functors

5.6.1 Composites of natural transformations Let $F : \mathscr{A} \to \mathscr{B}$ and $G : \mathscr{B} \to \mathscr{C}$ be functors. There is a composite functor $G \circ F : \mathscr{A} \to \mathscr{C}$ defined in the usual way by $G \circ F(A) = G(F(A))$. Similarly, let H, K and L be functors from $\mathscr{A} \to \mathscr{B}$ and $\alpha : H \to K$ and $\beta : K \to L$ be natural transformations. Recall that this means that for each object A of $\mathscr{A}$, $\alpha A : HA \to KA$ and $\beta A : KA \to LA$. Then as in 5.2.11, we define $\beta \circ \alpha : H \to L$ by

$$(\beta \circ \alpha)A = \beta A \circ \alpha A$$

There is a second way of composing natural transformations. See [Barr and Wells, 1999], Section 4.5.

5.6.2 Functors and natural transformations Things get more interesting when we mix functors and natural transformations. For example, suppose we have three categories $\mathscr{A}$, $\mathscr{B}$ and $\mathscr{C}$, four functors, two of them, $F, G : \mathscr{A} \to \mathscr{B}$ and the other two $H, K : \mathscr{B} \to \mathscr{C}$, and two natural transformations $\alpha : F \to G$ and $\beta : H \to K$. We picture this situation as follows:

$$\mathscr{A} \underset{G}{\overset{F}{\Downarrow \alpha}} \mathscr{B} \underset{K}{\overset{H}{\Downarrow \beta}} \mathscr{C} \tag{5.25}$$

5.6.3 Definition The natural transformation $\beta F : H \circ F \to K \circ F$ is defined by the formula $(\beta F)A = \beta(FA)$ for an object A of $\mathscr{A}$.

The notation $\beta(FA)$ means the component of the natural transformation β at the object FA. This is indeed an arrow from $H(F(A)) \to K(F(A))$ as required. To show that βF is natural requires showing that for an arrow $f : A \to A'$ of $\mathscr{A}$, the diagram

$$\begin{array}{ccc} H(F(A)) & \xrightarrow{\beta FA} & K(F(A)) \\ {\scriptstyle H(F(f))}\downarrow & & \downarrow{\scriptstyle K(F(f))} \\ H(F(A')) & \xrightarrow[\beta FA']{} & K(F(A')) \end{array} \tag{5.26}$$

commutes, but this is just the naturality diagram of β applied to the arrow $F(f) : F(A) \to F(A')$.

5.6.4 Definition The natural transformation $H\alpha : H \circ F \to H \circ G$ is defined by letting $(H\alpha)A = H(\alpha A)$ for an object A of $\mathscr{A}$, that is the value of H applied to the arrow αA.

To see that $H\alpha$ thus defined is natural requires showing that

$$\begin{array}{ccc} H(F(A)) & \xrightarrow{H(\alpha A)} & H(G(A)) \\ {\scriptstyle H(F(f))}\downarrow & & \downarrow{\scriptstyle H(G(f))} \\ H(F(A')) & \xrightarrow[H(\alpha A')]{} & H(G(A')) \end{array}$$

commutes. This diagram is obtained by applying the functor H to the naturality diagram of α. Since functors preserve commutative diagrams, the result follows.

Note that the proofs of naturality for βF and for $H\alpha$ are quite different. For example, the second requires that H be a functor, while the first works if F is merely an object function.

5.7 The Yoneda Lemma and universal elements

For an arbitrary category $\mathscr{C}$, the functors from $\mathscr{C}$ to **Set** are special because the hom functors $\mathrm{Hom}(C, -)$ for each object C of $\mathscr{C}$ are set-valued functors. In this section, we introduce the concept of representable functor, the Yoneda Lemma, and universal elements, all of which are based on these hom functors. These ideas have turned out to be fundamental tools for categorists. They are also closely connected with the concept of adjunction, to be discussed later (note Theorem 9.3.2 and Proposition 9.3.6).

If you are familiar with group theory, it may be illuminating to realize that representable functors are a generalization of the regular representation, and the Yoneda embedding is a generalization of Cayley's Theorem.

We have already considered set-valued functors as actions in Section 4.2.

5.7.1 Representable functors A functor from a category $\mathscr{C}$ to the category of sets (a **set-valued functor**) is said to be **representable** if it is naturally isomorphic to a hom functor; see 4.1.21. A covariant functor is representable if it is naturally isomorphic to $\mathrm{Hom}(C, -)$ for some object C of $\mathscr{C}$; in this case one says that C **represents** the functor. A contravariant functor is representable if it is naturally isomorphic to $\mathrm{Hom}(-, C)$ for some object C (and then C represents the contravariant functor).

We have already looked at one example of representable functor in some detail in 5.5.2, where we showed that the set-of-nodes functor for graphs is represented by the graph with one node and no arrows. The set-of-arrows functor is represented by the graph with two nodes and one arrow between them.

5.7.2 Example The identity functor on **Set** is represented by the terminal object; in other words, $\mathrm{Hom}_{\mathbf{Set}}(1, -)$ is naturally isomorphic to the identity functor. This can be described by saying that 'a set is its set of global elements.'

5.7.3 Example The arrow functor $A : \mathbf{Grf} \longrightarrow \mathbf{Set}$ of 4.1.10 is represented by the graph **2** which is pictured as

$$1 \xrightarrow{\ e\ } 2$$

5.7.4 Example The set of objects of a small category is 'essentially the same thing' as the set of global elements of the category (as an object of **Cat**). In other words, the set of objects functor is represented by the terminal object of **Cat**, which is the category with one object and one arrow.

The set of arrows of a small category is the object part of a functor that is represented by the category **2**, which is the graph **2** with the addition of two identity arrows.

5.7.5 The Yoneda embedding Let $\mathscr{C}$ be a category. There is a functor $Y : \mathscr{C}^{\mathrm{op}} \longrightarrow \mathbf{Func}(\mathscr{C}, \mathbf{Set})$, the **Yoneda functor**, defined as follows. Note that Y must take an object of $\mathscr{C}$ to a functor and an arrow of $\mathscr{C}$ to a natural transformation.

Y–1 For an object C of $\mathscr{C}$, $Y(C) = \mathrm{Hom}(C, -)$.

Y–2 If $f : D \longrightarrow C$ in $\mathscr{C}$ and A is an object of $\mathscr{C}$, then the component $Y(f)A$ of $Y(f) : \mathrm{Hom}(C, -) \longrightarrow \mathrm{Hom}(D, -)$ is $\mathrm{Hom}(f, A) : \mathrm{Hom}(C, A) \longrightarrow \mathrm{Hom}(D, A)$ (see 4.1.22).

Note that $Y(C)$ is a *covariant* hom functor and that $Y(f)A$ is a component of a *contravariant* hom functor.

To see that $Y(f)$ is a natural transformation requires checking that this diagram commutes for every arrow $k : A \longrightarrow B$ of $\mathscr{C}$:

$$\begin{array}{ccc} \mathrm{Hom}(D, A) & \xrightarrow{\mathrm{Hom}(D,k)} & \mathrm{Hom}(D, B) \\ {\scriptstyle Y(f)A}\uparrow & & \uparrow{\scriptstyle Y(f)B} \\ \mathrm{Hom}(C, A) & \xrightarrow[\mathrm{Hom}(C,k)]{} & \mathrm{Hom}(C, B) \end{array}$$

To see that it commutes, start with $h : C \longrightarrow A$, an arbitrary element of the lower left corner. The lower route takes this to $k \circ h$, then to $(k \circ h) \circ f$. The upper route takes it to $k \circ (h \circ f)$, so the fact that the diagram commutes is simply a statement of the associative law. In a monoid, that this diagram commutes is the statement that the function defined by left multiplying by a given element commutes with the function defined by right multiplying by another given element.

$Y(f) : \mathrm{Hom}(C, -) \longrightarrow \mathrm{Hom}(D, -)$ is the **induced natural transformation** corresponding to f.

The main theorem concerning Y is the following.

5.7.6 Theorem *$Y : \mathscr{C}^{\mathrm{op}} \longrightarrow \mathbf{Func}(\mathscr{C}, \mathbf{Set})$ is a full and faithful functor.*

The fact that Y is full and faithful is encapsulated in the following remarkable corollary.

5.7.7 Corollary *Every natural transformation*

$$\mathrm{Hom}(C,-) \longrightarrow \mathrm{Hom}(D,-)$$

*is given by composition with a unique arrow $D \longrightarrow C$. The natural transformation is an isomorphism if and only if the corresponding arrow $D \longrightarrow C$ is an isomorphism. In particular, if $F : \mathscr{C} \longrightarrow$ **Set** is represented by both C and D, then $C \cong D$.*

This means that you can construct an arrow in a category by constructing a natural transformation between hom functors. This is one of the most widely used techniques in category theory.

Proof. Theorem 5.7.6 is an immediate corollary of the Yoneda Lemma (5.7.11). We give a direct proof here. This proof is an excellent exercise in manipulating natural transformations and hom sets.

Let $f, g : D \longrightarrow C$ in $\mathscr{C}$. The component

$$Y(f)C : \mathrm{Hom}(C,C) \longrightarrow \mathrm{Hom}(D,C)$$

of the natural transformation $Y(f)$ at C takes id_C to f, and similarly $Y(g)C$ takes id_C to g. Thus if $f \neq g$, then $Y(f)C \neq Y(g)C$, so that $Y(f) \neq Y(g)$. Thus Y is faithful.

We must show that Y is full. Given $\phi : \mathrm{Hom}(C,-) \longrightarrow \mathrm{Hom}(D,-)$, we get the required $f : D \longrightarrow C$ by one of the basic tricks of category theory: we define $f = \phi C(\mathrm{id}_C)$. The component of ϕ at C is a function $\phi C : \mathrm{Hom}(C,C) \longrightarrow \mathrm{Hom}(D,C)$, so this definition makes sense.

To complete the proof, we must prove that if $k : C \longrightarrow A$ is any arrow of $\mathscr{C}$, then $\phi A(k) = k \circ f : D \longrightarrow A$. This follows from the fact that the following diagram commutes by naturality of ϕ:

$$\begin{array}{ccc} \mathrm{Hom}(C,C) & \xrightarrow{\mathrm{Hom}(C,k)} & \mathrm{Hom}(C,A) \\ \Big\downarrow{\scriptstyle \phi C} & & \Big\downarrow{\scriptstyle \phi A} \\ \mathrm{Hom}(D,C) & \xrightarrow[\mathrm{Hom}(D,k)]{} & \mathrm{Hom}(D,A) \end{array}$$

If you start in the northwest corner with id_C, the upper route takes you to $\phi A(k)$ in the southeast corner, whereas the lower route takes you to $k \circ f$, as required. □

5.7.8 By replacing $\mathscr{C}$ by $\mathscr{C}^{\mathrm{op}}$ in Theorem 5.7.6, we derive a second Yoneda functor

$$J : \mathscr{C} \longrightarrow \mathbf{Func}(\mathscr{C}^{\mathrm{op}}, \mathbf{Set})$$

which is also full and faithful. For an object C of $\mathscr{C}$, $J(C) = \mathrm{Hom}(-,C)$, the contravariant hom functor. If $f : C \longrightarrow D$ in $\mathscr{C}$ and A is an object of $\mathscr{C}$, then the component

$$J(f)A : \mathrm{Hom}(A,C) \longrightarrow \mathrm{Hom}(A,D)$$

of the natural transformation $J(f) : \mathrm{Hom}(-,C) \longrightarrow \mathrm{Hom}(-,D)$ is

$$\mathrm{Hom}(A,f) : \mathrm{Hom}(A,C) \longrightarrow \mathrm{Hom}(A,D)$$

The fact that J is full and faithful means that an arrow from A to B of $\mathscr{C}$ can be uniquely defined by giving a natural transformation from $\mathrm{Hom}(-,A)$ to $\mathrm{Hom}(-,B)$. This statement is the dual of Corollary 5.7.7. Such a natural transformation $\alpha : \mathrm{Hom}(-,A) \longrightarrow \mathrm{Hom}(-,B)$ has a component $\alpha T : \mathrm{Hom}(T,A) \longrightarrow \mathrm{Hom}(T,B)$ for each object T of $\mathscr{C}$. The effect of this is that you can define an arrow from A to B by giving a function $\alpha T : \mathrm{Hom}(T,A) \longrightarrow \mathrm{Hom}(T,B)$ for each object T which prescribes a variable element of B for each variable element of A (as described in 3.3.2), in such a way that for each $f : T' \longrightarrow T$, the diagram

$$\begin{array}{ccc} \mathrm{Hom}(T,A) & \xrightarrow{\alpha T} & \mathrm{Hom}(T,B) \\ \Big\downarrow{\scriptstyle \mathrm{Hom}(f,A)} & & \Big\downarrow{\scriptstyle \mathrm{Hom}(f,B)} \\ \mathrm{Hom}(T',A) & \xrightarrow[\alpha T']{} & \mathrm{Hom}(T',B) \end{array}$$

commutes. This can be summed up by saying, 'An arrow is induced by defining its value on each variable element of its domain, provided that the definition is natural with respect to change of parameters.'

5.7.9 Elements of a set-valued functor Corollary 5.7.7 says that any natural transformation from $\mathrm{Hom}(C,-)$ to $\mathrm{Hom}(D,-)$ is given by a unique arrow from D to C, that is, by an element of $\mathrm{Hom}(D,C)$, which is $\mathrm{Hom}(D,-)(C)$. Remarkably, the result remains true when $\mathrm{Hom}(D,-)$ is replaced by an arbitrary set-valued functor.

Suppose $F:\mathscr{C}\longrightarrow\mathbf{Set}$ is a functor and C is an object of $\mathscr{C}$. An element $c\in F(C)$ induces a natural transformation from the representable functor $\mathrm{Hom}(C,-)$ to F by the formula

$$f\mapsto F(f)(c) \tag{5.27}$$

That is, if $f:C\longrightarrow C'$ is an element of $\mathrm{Hom}(C,C')$, the definition of functor requires an induced function $F(f):F(C)\longrightarrow F(C')$ and this function can be evaluated at $c\in F(C)$.

5.7.10 Proposition *Formula (5.27) defines a natural transformation*

$$\mathrm{Hom}(C,-)\longrightarrow F$$

Proof. Let $\alpha C':\mathrm{Hom}(C,C')\longrightarrow F(C')$ take f to $F(f)(c)$ for $c\in F(C)$. We must show that for any $g:C'\longrightarrow B$,

$$\begin{array}{ccc} \mathrm{Hom}(C,C') & \xrightarrow{\alpha C'} & F(C') \\ {\scriptstyle \mathrm{Hom}(C,g)}\downarrow & & \downarrow{\scriptstyle F(g)} \\ \mathrm{Hom}(C,B) & \xrightarrow[\alpha B]{} & F(B) \end{array} \tag{5.28}$$

commutes. We have, for $f\in\mathrm{Hom}(C,C')$,

$$\begin{aligned} \alpha B(\mathrm{Hom}(C,g)(f)) &= \alpha B(g\circ f) = F(g\circ f)(c) \\ &= F(g)(F(f)(c)) = F(g)(\alpha C'(f)) \end{aligned}$$

as required.

5.7.11 Theorem (Yoneda Lemma) *Formula (5.27) defines a one to one correspondence between elements of $F(C)$ and natural transformations*

$$\mathrm{Hom}(C,-)\longrightarrow F$$

Proof. Suppose that c and c' are different elements of $F(C)$. Then the natural transformation corresponding to c takes id_C to c whereas the one corresponding to c' takes id_C to c'. Thus the mapping of the Yoneda Lemma is injective.

Suppose $\beta:\mathrm{Hom}(C,-)\longrightarrow F$ is a natural transformation. Then we have $\beta C:\mathrm{Hom}(C,C)\longrightarrow F(C)$. Let $c=\beta C(\mathrm{id}_C)\in F(C)$. For any $f:C\longrightarrow C'$, the naturality of β gives that

$$\beta C'(\mathrm{Hom}(C,f)(\mathrm{id}_C)) = F(f)(\beta C)(\mathrm{id}_C)$$

The left hand side is $\beta C'(f)$ and the right hand side is $F(f)(c)$. Thus β is the natural transformation given by Formula (5.27), so that the mapping of the Yoneda Lemma is surjective. □

5.7.12 Definition Let $F:\mathscr{C}\longrightarrow\mathbf{Set}$ be a functor and let c be an element of $F(C)$ for some object C of $\mathscr{C}$. If the natural transformation from $\mathrm{Hom}(C,-)$ to F induced by c is an isomorphism, then c is a **universal element** of F.

The existence of a universal element means that F is representable (see 5.7.1). The converse is also true because a natural isomorphism $\alpha:\mathrm{Hom}(C,-)\longrightarrow F$ is, from the Yoneda lemma, induced by a unique element c of $F(C)$ and by definition α is an isomorphism if and only if c is universal.

5.7.13 Example Let $D : \mathbf{Set} \to \mathbf{Set}$ be the functor for which for a set A, $D(A) = A \times A$, and for a function $f : A \to B$, $D(f) : A \times A \to B \times B$ is the function defined by $D(f)(a_1, a_2) = (f(a_1), f(a_2))$. Then D is represented by $\{1, 2\}$ (or any other two-element set). A universal object is $(1, 2)$ (or any other pair with distinct coordinates.) The natural isomorphism from $\mathrm{Hom}(\{1, 2\}, A)$ to $A \times A$ takes a function $f : \{1, 2\} \to A$ to the pair $(f(1), f(2))$. The inverse takes a pair (a_1, a_2) to the function f for which $f(i) = a_i$.

The unique element c given by the definition of universal element can be calculated using the following fact.

5.7.14 Proposition *Let $\alpha : \mathrm{Hom}(C, -) \to F$ be a natural isomorphism. The unique element $c \in F(C)$ inducing α is $\alpha C(\mathrm{id}_C)$.*

Proof. For an arbitrary $f : C \to C'$, $\alpha C'(f) = F(f)(\alpha C(\mathrm{id}_C))$ because this diagram must commute (chase id_C around the square):

$$\begin{array}{ccc} \mathrm{Hom}(C, C) & \xrightarrow{\alpha C} & F(C) \\ \Big\downarrow{\scriptstyle \mathrm{Hom}(C, f)} & & \Big\downarrow{\scriptstyle F(f)} \\ \mathrm{Hom}(C, C') & \xrightarrow[\alpha C']{} & F(C') \end{array}$$

Then, by Formula (5.27), $\alpha C(\mathrm{id}_C)$ must be the required unique element c. □

A detailed example of the use of this construction is in the proof of Proposition 6.2.14.

5.7.15 The definition of universal element can be reworded in elementary terms using Formula (5.27), as follows.

5.7.16 Proposition *Let $F : \mathscr{C} \to \mathbf{Set}$ be a functor, C an object of $\mathscr{C}$ and c an element of $F(C)$. Then c is a universal element of F if and only if for any object C' of $\mathscr{C}$ and any element $x \in F(C')$ there is a unique arrow $f : C \to C'$ of $\mathscr{C}$ for which $x = F(f)(c)$.*

Proof. If c is a universal element then the mapping (5.27) must be an isomorphism, hence every component must be bijective by Theorem 5.2.21. This immediately ensures the existence and uniqueness of the required arrow f. Conversely, the existence and uniqueness of f for each C' and $x \in F(C')$ means that there is a bijection $\alpha C' : \mathrm{Hom}(C, C') \to F(C')$ for every C' which takes $f : C \to C'$ to $F(f)(c)$. By Proposition 5.7.10, these are the components of a natural transformation, which is therefore a natural isomorphism by Theorem 5.2.21. □

In the case of a functor $F : \mathscr{C}^{\mathrm{op}} \to \mathbf{Set}$, c in $F(C)$ is a universal element if for any object C' of $\mathscr{C}$ and any element $x \in F(C')$ there is a unique arrow $f : C' \to C$ for which $x = F(f)(c)$.

5.7.17 Corollary *If $c \in F(C)$ and $c' \in F(C')$ are universal elements, then there is a unique isomorphism $f : C \to C'$ such that $F(f)(c) = c'$.*

The proof is omitted.

Universal elements are considered again in Proposition 9.3.6. The exposition in [Mac Lane, 1971] uses the concept of universal element (defined in the manner of the preceding proposition) as the central idea in discussing representable functors and adjunction.

6. Products and sums

This chapter introduces products, which are constructions allowing the definition of operations of arbitrary arity, and sums, which allow the specification of alternatives. In **Set**, the product is essentially the cartesian product, and the sum is disjoint union.

6.1 The product of two objects in a category

6.1.1 Definition If S and T are sets, the **cartesian product** $S \times T$ is the set of all ordered pairs with first coordinate in S and second coordinate in T; in other words, $S \times T = \{(s,t) \mid s \in S \text{ and } t \in T\}$. The coordinates are functions $\text{proj}_1 : S \times T \to S$ and $\text{proj}_2 : S \times T \to T$ called the **coordinate projections**, or simply **projections**.

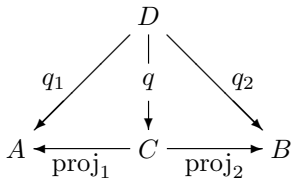
We give a specification of product of two objects in an arbitrary category which will have the cartesian product in **Set** as a special case. This specification is given in terms of the coordinate projections, motivated by these two facts:

(i) you know an element of $S \times T$ by knowing what its two coordinates are, and

(ii) given any element of S and any element of T, there is an element of $S \times T$ with the given element of S as first coordinate and the given element of T as second coordinate.

6.1.2 The product of two objects Let A and B be two objects in a category $\mathscr{C}$. By a (*not* the) **product** of A and B, we mean an object C *together with* arrows $\text{proj}_1 : C \to A$ and $\text{proj}_2 : C \to B$ that satisfy the following condition.

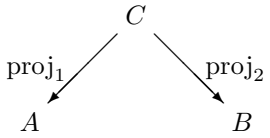
6.1.3 For any object D and arrows $q_1 : D \to A$ and $q_2 : D \to B$, there is a unique arrow $q : D \to C$:



(6.1)

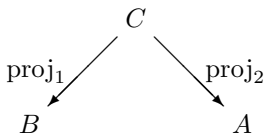
such that $\text{proj}_1 \circ q = q_1$ and $\text{proj}_2 \circ q = q_2$.

6.1.4 Product cones The specification above gives the product as C together with proj_1 and proj_2. The corresponding diagram



(6.2)

is called a **product diagram** or **product cone**, and the arrows proj_i are called the **projections**. These projections are indexed by the set $\{1,2\}$. The **base** of the cone is the diagram $D : \mathscr{I} \to \mathscr{C}$, where $\mathscr{I}$ is the discrete graph with two nodes 1 and 2 and no arrows. This amounts to saying that the base of the cone is the ordered pair (A,B). The diagram



(6.3)

is regarded as a different product cone since its base is the diagram D with $D(1) = B$ and $D(2) = A$.

By a type of synecdoche, one often says that an object (such as C above) 'is' a product of two other objects (here A and B), leaving the projections implicit, but the projections are nevertheless part of the structure we call 'product'.

Products can be based on discrete graphs with other shape graphs, having more elements (Section 6.3) or having other sets of nodes, for example attributes of a data base such as $\{\texttt{NAME}, \texttt{SALARY}\}$ (see 6.3.13). In this case the projections would be $\text{proj}_{\text{NAME}}$ and $\text{proj}_{\text{SALARY}}$.

The existence of the unique arrow q with the property given in 6.1.3 is called the **universal mapping property** of the product. Any object construction which is defined up to a unique isomorphism (see Theorem 6.2.2) in terms of arrows into or out of it is often said to be defined by a universal mapping property.

6.1.5 Products in Set If S and T are sets, then the cartesian product $S \times T$, together with the coordinate functions discussed in 6.1.1, is indeed a product of S and T in **Set**. For suppose we have a set V and two functions $q_1 : V \longrightarrow S$ and $q_2 : V \longrightarrow T$. The function $q : V \longrightarrow S \times T$ defined by

$$q(v) = (q_1(v), q_2(v))$$

for $v \in V$ is the unique function satisfying 6.1.3. Since $\text{proj}_i(q(v)) = q_i(v)$ by definition, q makes (6.1) commute with $U = S \times T$, and it must be the only such function since the commutativity of (6.1) determines that its value at v *must* be $(q_1(v), q_2(v))$.

We discuss products in **Rel** and in **Pfn** in 6.4.7.

6.1.6 Products in categories of sets with structure In many, but not all, categories of sets with structure, the product can be constructed by endowing the product set with the structure in an obvious way.

6.1.7 Example If S and T are semigroups, then we can make $S \times T$ into a semigroup by defining the multiplication

$$(s_1, t_1)(s_2, t_2) = (s_1 s_2, t_1 t_2)$$

We verify associativity by the calculation

$$\begin{aligned} [(s_1, t_1)(s_2, t_2)](s_3, t_3) &= (s_1 s_2, t_1 t_2)(s_3, t_3) \\ &= ((s_1 s_2)s_3, (t_1 t_2)t_3) \\ &= (s_1(s_2 s_3), t_1(t_2 t_3)) \\ &= (s_1, t_1)(s_2 s_3, t_2 t_3) \\ &= (s_1, t_1)[(s_2, t_2)(s_3, t_3)] \end{aligned} \tag{6.4}$$

Furthermore, this structure together with the coordinate projections satisfies the definition of product in the category of semigroups. To see this requires showing two things:

(a) The arrows $\text{proj}_1 : S \times T \longrightarrow S$ and $\text{proj}_2 : S \times T \longrightarrow T$ are homomorphisms of semigroups.

(b) If q_1 and q_2 are semigroup homomorphisms, then so is the arrow q determined by 6.1.3.

It is necessary to show both because the definition of product in a category $\mathscr{C}$ requires that the arrows occurring in Diagram (6.1) be arrows of the category, in this case, **Sem**.

Requirement (a) follows from this calculation:

$$\begin{aligned} \text{proj}_1((s_1, t_1)(s_2, t_2)) &= \text{proj}_1(s_1 s_2, t_1 t_2) = s_1 s_2 \\ &= \text{proj}_1(s_1, t_1)\,\text{proj}_1(s_2, t_2) \end{aligned}$$

and similarly for proj_2.

As for requirement (b), let R be another semigroup and $q_1 : R \longrightarrow S$ and $q_2 : R \longrightarrow T$ be homomorphisms. Then

$$\begin{aligned} \langle q_1, q_2\rangle(r_1 r_2) &= (q_1(r_1 r_2), q_2(r_1 r_2)) = (q_1(r_1)q_1(r_2), q_2(r_1)q_2(r_2)) \\ &= (q_1(r_1), q_2(r_1))(q_1(r_2), q_2(r_2)) \\ &= \langle q_1, q_2\rangle(r_1)\langle q_1, q_2\rangle(r_2) \end{aligned}$$

A construction for products similar to that for semigroups works for most other categories of sets with structure. Also, the product of categories as defined in 2.6.6 is the product of the categories in **Cat**. One example of a category of sets with structure which lacks products is the category of fields.

6.1.8 Example Let $\mathscr{G}$ and $\mathscr{H}$ be two graphs. The product $\mathscr{G} \times \mathscr{H}$ in the category of graphs and homomorphisms is defined as follows: $(\mathscr{G} \times \mathscr{H})_0 = G_0 \times H_0$. An arrow from (g, h) to (g', h') is a pair (a, b) with $a : g \longrightarrow g'$ in $\mathscr{G}$ and $b : h \longrightarrow h'$ in $\mathscr{H}$. The projections are the usual first and second projections.

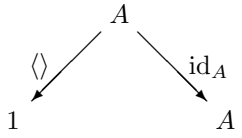
6.1.9 Products in posets We have already seen in 2.3.1 that any poset (partially ordered set) has a corresponding category structure $C(P)$. Let P be a poset and x and y two objects of $C(P)$ (that is, elements of P). Let us see what, if anything, is their product. A product must be an element z together with a pair of arrows $z \longrightarrow x$ and $z \longrightarrow y$, which is just another way of saying that $z \leq x$ and $z \leq y$. The definition of product also requires that for any $w \in P$, given an arrow $w \longrightarrow x$ and one $w \longrightarrow y$, there is an arrow $w \longrightarrow z$.

This translates to

$$w \leq x \text{ and } w \leq y \text{ implies } w \leq z$$

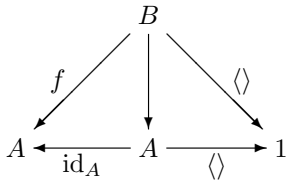
which, together with the fact that $z \leq x$ and $z \leq y$, characterizes z as the **infimum** of x and y, often denoted $x \wedge y$. Thus the existence of products in such a category is equivalent to the existence of infimums. In particular, we see that products generalize a well-known construction in posets. Note that a poset that lacks infimums provides an easy example of a category without products.

6.1.10 Proposition *If A is an object in a category with a terminal object 1, then*



is a product diagram.

Proof. A cone over A and 1 has to have this form, where $f : B \longrightarrow A$ is any arrow.



Clearly the only possible arrow in the middle is f. □

6.2 Notation for and properties of products

6.2.1 Consider sets $S = \{1, 2, 3\}$, $T = \{1, 2\}$ and $U = \{1, 2, 3, 4, 5, 6\}$. Define $\text{proj}_1 : U \longrightarrow S$ and $\text{proj}_2 : U \longrightarrow T$ by this table:

u	$\text{proj}_1(u)$	$\text{proj}_2(u)$
1	2	1
2	1	1
3	3	1
4	2	2
5	1	2
6	3	2

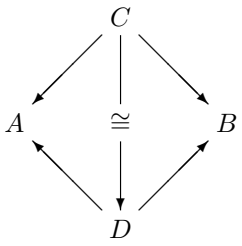
Since the middle and right columns give every possible combination of a number 1, 2 or 3 followed by a number 1 or 2, it follows that U, together with proj_1 and proj_2, is a product of S and T. For example, if $q_1 : V \longrightarrow S$ and $q_2 : V \longrightarrow T$ are given functions and $q_1(v) = 1$, $q_2(v) = 2$ for some v in V, then the unique function $q : V \longrightarrow U$ satisfying 6.1.3 must take v to 5.

In effect, proj_1 and proj_2 code the ordered pairs in $S \times T$ into the set U. As you can see, any choice of a six-element set U and any choice of proj_1 and proj_2 which gives a different element of U for each ordered pair in $S \times T$ gives a product of S and T.

This example shows that the categorical concept of product gives a more general construction than the cartesian product construction for sets. One cannot talk about 'the' product of two objects, but only of 'a' product. However, the following theorem says that two products of the same two objects are isomorphic in a strong sense.

6.2.2 Theorem *Let $\mathscr{C}$ be a category and let A and B be two objects of $\mathscr{C}$. Suppose*

are both product diagrams. Then there is an arrow, and only one, from C to D such that



(6.5)

commutes and this arrow is an isomorphism.

The proof we give is quite typical of the kind of reasoning common in category theory and is worth studying, although not necessarily on first reading.

Proof. Let the projections be $p_1 : C \longrightarrow A$, $p_2 : C \longrightarrow B$, $q_1 : D \longrightarrow A$ and $q_2 : D \longrightarrow B$. In accordance with 6.1.3, there are unique arrows $p : C \longrightarrow D$ and $q : D \longrightarrow C$ for which

$$\begin{array}{rcl} p_1 \circ q & = & q_1 \\ p_2 \circ q & = & q_2 \\ q_1 \circ p & = & p_1 \\ q_2 \circ p & = & p_2 \end{array} \tag{6.6}$$

Thus we already know there is exactly one arrow (namely p) making Diagram (6.5) commute; all that is left to prove is that p is an isomorphism (with inverse q).

The arrow $q \circ p : C \longrightarrow C$ satisfies

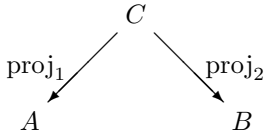
$$p_1 \circ q \circ p = q_1 \circ p = p_1 = p_1 \circ \mathrm{id}_C$$

$$p_2 \circ q \circ p = q_2 \circ p = p_2 = p_2 \circ \mathrm{id}_C$$

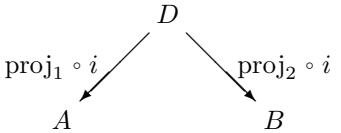
and by the uniqueness part of 6.1.3, it follows that $q \circ p = \mathrm{id}_C$. If we exchange the p's and q's, we similarly conclude that $p \circ q = \mathrm{id}_D$ and hence that p and q are isomorphisms which are inverse to each other. □

The following proposition is a converse to Theorem 6.2.2. Its proof is left as an exercise.

6.2.3 Proposition *Let*



be a product diagram, and suppose that an object D is isomorphic to C by an isomorphism $i : D \longrightarrow C$. Then



is a product diagram.

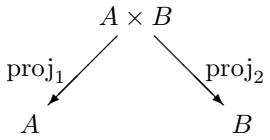
6.2.4 Categorists specify the product of two sets by saying that all they care about an element of the product is what its first coordinate is and what its second coordinate is. Theorem 6.2.2 says that two structures satisfying this specification are isomorphic in a unique way.

The name '(s,t)' represents the element of the product with first coordinate s and second coordinate t. In a different realization of the product, '(s,t)' represents the element of *that* product with first coordinate s and second coordinate t. The isomorphism of Theorem 6.2.2 maps the representation in the first realization of the product into a representation in the second. Moreover, the universal property of product says that any name '(s,t)' with $s \in S$ and $t \in T$ represents an element of the product: it is the unique element $x \in S \times T$ with $\mathrm{proj}_1(x) = s$ and $\mathrm{proj}_2(x) = t$.

In traditional approaches to foundations, the concept of ordered pair (hence the product of two sets) is defined by giving a specific model (or what a computer scientist might call an implementation) of the specification. Such a definition makes the product absolutely unique instead of unique up to an isomorphism. We recommend that the reader read the discussion of this point in [Halmos, 1960], Section 6, who gives a beautiful discussion of (what in present day language we call) the difference between a specification and an implementation.

In categories other than sets there may well be no standard implementation of products, so the specification given is necessary. In Chapter 11, we will discuss a category known as the category of modest sets in which any construction requires the choice of a bijection between $\mathbf{N}$ and $\mathbf{N} \times \mathbf{N}$. There are many such, and there is no particular reason to choose one over another.

6.2.5 Notation for products It is customary to denote a product of objects A and B of a category as $A \times B$. Precisely, the name $A \times B$ applied to an object means there is a product diagram



(6.7)

Using the name '$A \times B$' implies that there are specific, but unnamed, projections given for the product structure.

If $A = B$, one writes $A \times A = A^2$ and calls it the **cartesian square** of A.

The notations $A \times B$ and A^2 may be ambiguous, but because of Theorem 6.2.2, it does not matter *for categorical purposes* which product the symbol refers to.

> Even in the category of sets, you do not really know which set $A \times B$ is unless you pick a specific definition of ordered pair, and the average mathematician does not normally need to give any thought to the definition because what really matters is the universal property that says that an ordered pair is determined by its values under the projections.

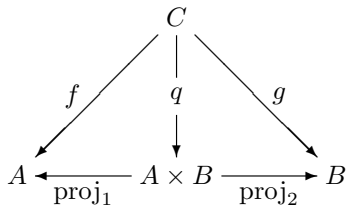
6.2.6 Binary operations A **binary operation** on a set S is a function from $S \times S$ to S. An example is addition on the natural numbers, which is a function $+ : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$. This and other familiar binary operations are usually written in infix notation; one writes $3+5 = 8$, for example, instead of $+(3,5) = 8$. In mathematics texts, the value of an arbitrary binary operation m at a pair (x,y) is commonly denoted xy, without any symbol at all.

Using the concept of categorical product, we can now define the concept of binary operation on any object S of any category provided only that there is a product $S \times S$: a **binary operation** on S is an arrow $S \times S \to S$.

The associative law $(xy)z = x(yz)$ can be described using a commutative diagram as illustrated in 5.1.11. In that section, the diagram is a diagram in **Set**, but now it has a meaning in any category with products. (The meaning of expressions such as $\mathrm{mult} \times S$ in arbitrary categories is given in 6.2.17 below.)

The more general concept of function of two variables can now be defined in a categorical setting: an arrow $f : A \times B \to C$ can be thought of as the categorical version of a function of two variables. This has the consequence that a categorist thinks of a function such as $f : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ defined by $f(x,y) = x^2 + y^2$ as a function of *one* variable, but that variable is a structured variable (an ordered pair). The notation we have been using would suggest that one write this as $f((x,y))$ instead of $f(x,y)$, but no one does.

6.2.7 Suppose we are given a product diagram (6.7). For each pair of arrows $f : C \longrightarrow A$ and $g : C \longrightarrow B$ requirement 6.1.3 produces a unique $q : C \longrightarrow A \times B$ making the following diagram commute.



(6.8)

In other words, it produces a function

$$\pi C : \mathrm{Hom}_{\mathscr{C}}(C, A) \times \mathrm{Hom}_{\mathscr{C}}(C, B) \longrightarrow \mathrm{Hom}_{\mathscr{C}}(C, A \times B)$$

Thus $q = \pi C(f, g)$.

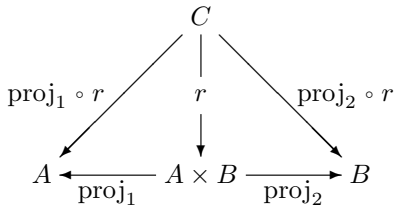
6.2.8 Proposition *The function πC is a bijection.*

Proof. πC is injective, since if (f, g) and (f', g') are elements of

$$\mathrm{Hom}_{\mathscr{C}}(C, A) \times \mathrm{Hom}_{\mathscr{C}}(C, B)$$

both of which produce the same arrow q making Diagram (6.8) commute, then $f = \mathrm{proj}_1 \circ q = f'$, and similarly $g = g'$.

It is also surjective, since if $r : C \longrightarrow A \times B$ is any arrow of $\mathscr{C}$, then it makes



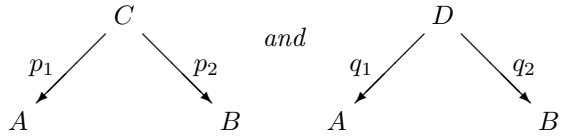
(6.9)

commute, and so is the image of the pair $(\mathrm{proj}_1 \circ r, \mathrm{proj}_2 \circ r)$ under πC. □

6.2.9 It is customary to write $\langle f, g\rangle$ for $\pi C(f, g)$. The *arrow* $\langle f, g\rangle$ internally represents the *pair* of arrows (f, g) of the category $\mathscr{C}$. Proposition 6.2.8 says that the representation is good in the sense that $\langle f, g\rangle$ and (f, g) each determine the other. Proposition 6.2.14 below says that the notation $\langle f, g\rangle$ is compatible with composition.

6.2.10 In the case of two products for the same pair of objects, the isomorphism of 6.2.2 translates the arrow named $\langle f, g\rangle$ for one product into the arrow named $\langle f, g\rangle$ for the other, in the following precise sense.

6.2.11 Proposition *Suppose*



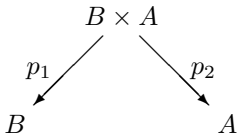
are two product diagrams and $\phi : C \longrightarrow D$ is the unique isomorphism given by Theorem 6.2.2. Let $f : E \longrightarrow A$ and $g : E \longrightarrow B$ be given and let $u : E \longrightarrow C$, $v : E \longrightarrow D$ be the unique arrows for which $p_1 \circ u = q_1 \circ v = f$ and $p_2 \circ u = q_2 \circ v = g$. Then $\phi \circ u = v$.

Note that in the statement of the theorem, both u and v could be called '$\langle f, g\rangle$', as described in 6.2.9. The ambiguity occurs because the pair notation does not name which product of A and B is being used. It is rare in practice to have two different products of the same two objects under consideration at the same time.

Proof. By 6.2.2, $p_i = q_i \circ \phi$, $i = 1, 2$. Using this, we have $q_1 \circ \phi \circ u = p_1 \circ u = f$ and similarly $q_2 \circ \phi \circ u = g$. Since v is the unique arrow which makes $q_1 \circ v = u$ and $q_2 \circ v = g$, it follows that $\phi \circ u = v$. □

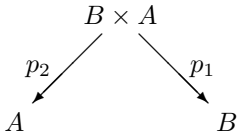
This theorem provides another point of view concerning elements (s, t) of a product $S \times T$. As described in 3.2.3, the element s may be represented by an arrow $s : 1 \longrightarrow S$, and similarly t by $t : 1 \longrightarrow T$. Then the arrow $\langle s, t\rangle : 1 \longrightarrow S \times T$ represents the ordered pair (s, t) whichever realization of $S \times T$ is chosen.

6.2.12 The switch map Our notation $A \times B$ means that $A \times B$ is the vertex of a product cone with base the discrete diagram D with $D(1) = A$ and $D(2) = B$. Then $B \times A$ denotes the product given by the diagram



(6.10)

where we use p_1 and p_2 to avoid confusing them with the arrows proj_1 and proj_2 of Diagram (6.7). (Of course, this is an *ad hoc* solution. If one had to deal with this situation a lot it would be necessary to introduce notation such as $\mathrm{proj}_1^{A,B}$ and $\mathrm{proj}_1^{B,A}$.) Then this is a product diagram:



(6.11)

It follows from Theorem 6.2.2 that there is an isomorphism $\langle p_2, p_1\rangle : B \times A \longrightarrow A \times B$ (called the **switch map**) that commutes with the projections. Its inverse is $\langle \mathrm{proj}_2, \mathrm{proj}_1\rangle : A \times B \longrightarrow B \times A$.

6.2.13 To show that the notation $\langle q_1, q_2\rangle$ is compatible with composition, we will show that the arrows πC defined in 6.2.7 are the components of a natural isomorphism. To state this claim formally, we need to make $\mathrm{Hom}_{\mathscr{C}}(C, A) \times \mathrm{Hom}_{\mathscr{C}}(C, B)$ into a functor. This is analogous to the definition of the contravariant hom functor. A and B are fixed and the varying object is C, so we define the functor $\mathrm{Hom}_{\mathscr{C}}(-, A) \times \mathrm{Hom}_{\mathscr{C}}(-, B)$ as follows:

(i) $[\mathrm{Hom}_{\mathscr{C}}(-, A) \times \mathrm{Hom}_{\mathscr{C}}(-, B)](C) = \mathrm{Hom}_{\mathscr{C}}(C, A) \times \mathrm{Hom}_{\mathscr{C}}(C, B)$, the set of pairs (g, h) of arrows $g : C \longrightarrow A$ and $h : C \longrightarrow B$.

(ii) For $f : D \longrightarrow C$, let $\mathrm{Hom}_{\mathscr{C}}(f, A) \times \mathrm{Hom}_{\mathscr{C}}(f, B)$ be the arrow

$$\mathrm{Hom}_{\mathscr{C}}(C, A) \times \mathrm{Hom}_{\mathscr{C}}(C, B) \longrightarrow \mathrm{Hom}_{\mathscr{C}}(D, A) \times \mathrm{Hom}_{\mathscr{C}}(D, B)$$

that takes a pair (g, h) to $(g \circ f, h \circ f)$.

Now we can state the proposition.

6.2.14 Proposition

$$\pi C : \mathrm{Hom}_{\mathscr{C}}(C, A) \times \mathrm{Hom}_{\mathscr{C}}(C, B) \longrightarrow \mathrm{Hom}_{\mathscr{C}}(C, A \times B)$$

constitutes a natural isomorphism

$$\pi : \mathrm{Hom}(-, A) \times \mathrm{Hom}(-, B) \longrightarrow \mathrm{Hom}(-, A \times B)$$

Proof. We give a proof in detail of this proposition here, but you may want to skip it on first reading, or for that matter on fifteenth reading. We will not always give proofs of similar statements later (of which there are many).

Let $P_{A,B}$ denote the functor $\mathrm{Hom}(-, A) \times \mathrm{Hom}(-, B)$. The projections $p_1 : A \times B \longrightarrow A$ and $p_2 : A \times B \longrightarrow B$ from the product form a pair $(p_1, p_2) \in P_{A,B}(A \times B)$.

6.2.15 Lemma *The pair (p_1, p_2) is a universal element for $P_{A,B}$.*

Proof. The pair fits the requirements of Proposition 5.7.16 by definition of product: if $(q_1, q_2) \in P_{A,B}(V)$, in other words if $q_1 : C \to A$ and $q_2 : C \to B$, there is a unique arrow $q : C \to A \times B$ such that $p_i \circ q = q_i$ for $i = 1, 2$. By 6.2.13(ii), $P_{A,B}(q)(p_1, p_2) = (q_1, q_2)$ as required.

Note that this gives an immediate proof of Theorem 6.2.2. See 9.2.3 for another point of view concerning (p_1, p_2).

Continuing the proof of Proposition 6.2.14, it follows from Equation (5.27) that the natural isomorphism α from $\mathrm{Hom}_{\mathscr{C}}(-, A \times B)$ to $P_{A,B}$ induced by this universal element takes $q : V \to A \times B$ to $(p_1 \circ q, p_2 \circ q)$, which is $P_{A,B}(q)(p_1, p_2)$. Then by definition of π we have that $\alpha C = (\pi C)^{-1}$, so that π is the inverse of a natural isomorphism and so is a natural isomorphism. □

It is not hard to give a direct proof of Proposition 6.2.14 using Proposition 6.2.8 and the definition of natural transformation. That definition requires that the following diagram commute for each arrow $f : C \to D$:

$$\begin{array}{ccc} \mathrm{Hom}(C,A) \times \mathrm{Hom}(C,B) & \xrightarrow{\pi D} & \mathrm{Hom}(C, A \times B) \\ \uparrow & & \uparrow \\ \mathrm{Hom}(D,A) \times \mathrm{Hom}(D,B) & \xrightarrow[\pi D]{} & \mathrm{Hom}(D, A \times B) \end{array} \tag{6.12}$$

In this diagram, the left arrow is defined as in Section 6.2.13 and the right arrow is defined as in Section 4.1.22.

6.2.16 If $f : C \to D$ and $q_1 : D \to A$ and $q_2 : D \to B$ determine $\langle q_1, q_2 \rangle : D \to A \times B$, then the commutativity of (6.12) says exactly that

$$\langle q_1 \circ f, q_2 \circ f \rangle = \langle q_1, q_2 \rangle \circ f \tag{6.13}$$

In this sense, the $\langle f, g \rangle$ notation is compatible with composition.

Category theorists say that the single arrow $\langle q_1, q_2 \rangle$ is the *internal* pair of arrows with first coordinate q_1 and second coordinate q_2. The idea behind the word 'internal' is that the category $\mathscr{C}$ is the workspace; inside that workspace the arrow $\langle q_1, q_2 \rangle$ is the pair (q_1, q_2).

When you think of $\mathscr{C}$ as a structure and look at it from the outside, you would say that the arrow q *represents* the *external* pair of arrows (q_1, q_2).

6.2.17 The cartesian product of arrows The cartesian product construction for functions in sets can also be given a categorical definition. Suppose that $f : S \to S'$ and $g : T \to T'$ are given. Then the composite arrows $f \circ \mathrm{proj}_1 : S \times T \to S'$ and $g \circ \mathrm{proj}_2 : S \times T \to T'$ induce, by the definition of product, an arrow denoted $f \times g : S \times T \to S' \times T'$ such that

$$\begin{array}{ccccc} S & \xleftarrow{\mathrm{proj}_1} & S \times T & \xrightarrow{\mathrm{proj}_2} & T \\ \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle f \times g} & & \downarrow{\scriptstyle g} \\ S' & \xleftarrow[\mathrm{proj}_1]{} & S' \times T' & \xrightarrow[\mathrm{proj}_2]{} & T' \end{array} \tag{6.14}$$

commutes. Thus $f \times g = \langle f \circ \mathrm{proj}_1, g \circ \mathrm{proj}_2 \rangle$. It is characterized by the properties

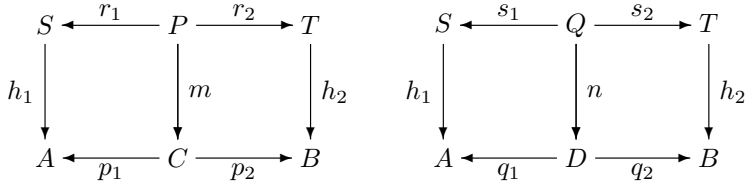
$$\mathrm{proj}_1 \circ (f \times g) = f \circ \mathrm{proj}_1; \quad \mathrm{proj}_2 \circ (f \times g) = g \circ \mathrm{proj}_2$$

Note that we use proj_1 and proj_2 for the product projections among different objects. This is standard and rarely causes confusion since the domains and codomains of the other arrows determine them. We will later call them p_1 and p_2, except for emphasis.

When one of the arrows f or g is an identity arrow, say $f = \mathrm{id}_S$, it is customary to write $S \times g$ for $\mathrm{id}_S \times g$.

An invariance theorem similar to Proposition 6.2.11 is true of cartesian products of functions.

6.2.18 Proposition *Suppose the top and bottom lines of each diagram below are product cones, and that m and n are the unique arrows making the diagrams commute. Let $\psi : P \longrightarrow Q$ and $\phi : C \longrightarrow D$ be the unique isomorphisms given by Theorem 6.2.2. Then $\phi \circ m = n \circ \psi$.*



Proof. For $i = 1, 2$,

$$\begin{aligned} q_i \circ \phi \circ m \circ \psi^{-1} &= p_i \circ m \circ \psi^{-1} \\ &= h_i \circ r_i \circ \psi^{-1} \\ &= h_i \circ s_i \circ \psi \circ \psi^{-1} \\ &= h_i \circ s_i \end{aligned} \tag{6.15}$$

The first equality is the property of ϕ given by Theorem 6.2.2, the second by definition of product applied to C, and the third is the defining property of ψ. It follows that $\phi \circ m \circ \psi^{-1}$ is the unique arrow determined by $h_1 \circ s_1 : Q \longrightarrow A$ and $h_2 \circ s_2 : Q \longrightarrow B$ and the fact that D is a product. But n is that arrow, so that $\phi \circ m \circ \psi^{-1} = n$, whence the theorem. □

6.2.19 Products and composition Let $\mathscr{C}$ be a category with products, and suppose $f_i : A_i \longrightarrow B_i$ and $g_i : B_i \longrightarrow C_i$ for $i = 1, 2$, so that $g_1 \circ f_1$ and $g_2 \circ f_2$ are defined. Then

$$(g_1 \circ f_1) \times (g_2 \circ f_2) = (g_1 \times g_2) \circ (f_1 \times f_2) : A_1 \times A_2 \longrightarrow C_1 \times C_2 \tag{6.16}$$

This follows from the fact that $(g_1 \circ f_1) \times (g_2 \circ f_2)$ is the unique arrow such that

$$\mathrm{proj}_1 \circ ((g_1 \circ f_1) \times (g_2 \circ f_2)) = (g_1 \circ f_1) \circ \mathrm{proj}_1$$

and

$$\mathrm{proj}_2 \circ ((g_1 \circ f_1) \times (g_2 \circ f_2)) = (g_2 \circ f_2) \circ \mathrm{proj}_2$$

Because Diagram (6.14) commutes, we have

$$\begin{aligned} \mathrm{proj}_1 \circ (g_1 \times g_2) \circ (f_1 \times f_2) &= g_1 \circ \mathrm{proj}_1 \circ (f_1 \times f_2) \\ &= g_1 \circ f_1 \circ \mathrm{proj}_1 : A_1 \times A_2 \longrightarrow C_1 \end{aligned}$$

and similarly

$$\mathrm{proj}_2 \circ (g_1 \times g_2) \circ (f_1 \times f_2) = g_2 \circ f_2 \circ \mathrm{proj}_2 : A_1 \times A_2 \longrightarrow C_2$$

so the result follows from the uniqueness of $(g_1 \circ f_1) \times (g_2 \circ f_2)$.

This fact allows us to see that the product of two objects is the value of a functor. Define $- \times - : \mathscr{C} \times \mathscr{C} \longrightarrow \mathscr{C}$ as follows: choose, for each pair A and B of $\mathscr{C}$, a product object $A \times B$, and let $(- \times -)(A, B) = A \times B$ and $(- \times -)(f, g) = f \times g$. Equation (6.16) shows that this mapping preserves composition and identities.

Another useful equation is the following, where we assume $f : A \longrightarrow C$, $g : B \longrightarrow D$, $u : X \longrightarrow A$ and $v : X \longrightarrow B$.

$$(f \times g) \circ \langle u, v \rangle = \langle f \circ u, g \circ v \rangle \tag{6.17}$$

The proof is left as an exercise.

6.2.20 Proposition *Let $\mathscr{C}$ and $\mathscr{D}$ be any categories. If $\mathscr{D}$ has products, then the functor category* **Func**$(\mathscr{C}, \mathscr{D})$ *also has products.*

Proof. The product is constructed by constructing the product at each value $F(C)$ and $G(C)$. Precisely, given two functors $F, G : \mathscr{C} \to \mathscr{D}$, the product in $\mathbf{Func}(\mathscr{C}, \mathscr{D})$ of F and G is the functor $F \times G$ defined as follows. For an object C of $\mathscr{C}$, $(F \times G)(C) = F(C) \times G(C)$, the product of the sets $F(C)$ and $G(C)$ in $\mathscr{D}$. For an arrow $f : C \to D$, $(F \times G)(f) = F(f) \times G(f)$, the product of the arrows as defined in 6.2.17. The projection $\pi_1 : F \times G \to F$ is the natural transformation whose component at C is $\pi_1 C = p_i : F(C) \times G(C) \to F(C)$, the product projection in $\mathscr{D}$. For any $f : C \to D$ in $\mathscr{C}$, the diagrams

$$\begin{array}{ccc} FC \times GC & \xrightarrow{p_1} & FC \\ {\scriptstyle F(f)\times G(f)}\downarrow & & \downarrow{\scriptstyle F(f)} \\ FD \times GD & \xrightarrow[p_1]{} & FD \end{array} \qquad \begin{array}{ccc} FC \times GC & \xrightarrow{p_2} & GC \\ {\scriptstyle F(f)\times G(f)}\downarrow & & \downarrow{\scriptstyle G(f)} \\ FD \times GD & \xrightarrow[p_2]{} & GD \end{array} \tag{6.18}$$

commute by definition of $F(f) \times G(f)$ (6.2.17), so that π_1 and π_2 are natural transformations as required.

Given natural transformations $\alpha : H \to F$ and $\beta : H \to G$, we must define $\langle \alpha, \beta \rangle : H \to F \times G$. For an object C, the component $\langle \alpha, \beta \rangle C = \langle \alpha C, \beta C \rangle : H(C) \to F(C) \times \beta(C)$. To see that $\langle \alpha, \beta \rangle$ is a natural transformation, we must show that for any arrow $f : C \to D$, this diagram commutes:

$$\begin{array}{ccc} H(C) & \xrightarrow{\langle \alpha, \beta \rangle C} & F(C) \times G(C) \\ {\scriptstyle H(f)}\downarrow & & \downarrow{\scriptstyle F(f)\times G(f)} \\ H(D) & \xrightarrow[\langle \alpha, \beta \rangle D]{} & F(D) \times G(D) \end{array} \tag{6.19}$$

This follows from the following calculation:

$$\begin{aligned} (F(f) \times G(f)) \circ \langle \alpha, \beta \rangle C &= (F(f) \times G(f)) \circ \langle \alpha C, \beta C \rangle \\ &= \langle F(f) \circ \alpha C, G(f) \circ \beta C \rangle \\ &= \langle \alpha D \circ H(f), \beta D \circ H(f) \rangle \\ &= \langle \alpha D, \beta D \rangle \circ H(f) \\ &= \langle \alpha, \beta \rangle D \circ H(f) \end{aligned}$$

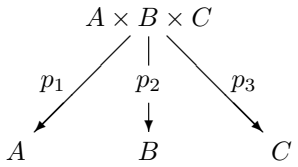
in which the first and last equalities are by definition of $\langle \alpha, \beta \rangle$, the second is by Equation (6.17), the third because α and β are natural transformations, and the fourth by Equation (6.13). □

Categorists say that this construction shows that $\mathbf{Func}(\mathscr{C}, \mathscr{D})$ has 'pointwise products'. This is the common terminology, but it might be better to say it has 'objectwise products'.

6.3 Finite products

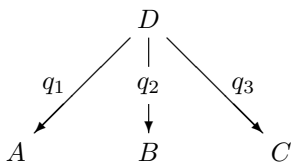
Products of two objects, as discussed in the preceding sections, are called **binary products**. We can define products of more than two objects by an obvious modification of the definition.

For example, if A, B and C are three objects of a category, a product of them is an object $A \times B \times C$ *together with* three arrows:



(6.20)

for which, given any other diagram



there exists a unique arrow $q = \langle q_1, q_2, q_3\rangle : D \longrightarrow A \times B \times C$ such that $p_i \circ q = q_i$, $i = 1,\ 2,\ 3$. A diagram of the form (6.20) is called a **ternary product diagram** (or ternary product cone). The general definition of product follows the same pattern.

6.3.1 Definition A **product** of a list $A_1, A_2, \ldots, A_n$ of objects (not necessarily distinct) of a category is an object V together with arrows $p_i : A \longrightarrow A_i$, for $i = 1, \ldots, n$, with the property that given any object B and arrows $f_i : B \longrightarrow A_i$, $i = 1, \ldots, n$, there is a unique arrow $\langle f_1, f_2, \ldots, f_n\rangle : B \longrightarrow A$ for which $p_i \circ \langle f_1, f_2, \ldots, f_n\rangle = f_i$, $i = 1, \ldots, n$.

A product of such a list $A_1, A_2, \ldots, A_n$ is called an **n-ary product** when it is necessary to specify the number of factors. Such a product may be denoted $A_1 \times A_2 \times \cdots \times A_n$ or $\prod_{i=1}^{n} A_i$.

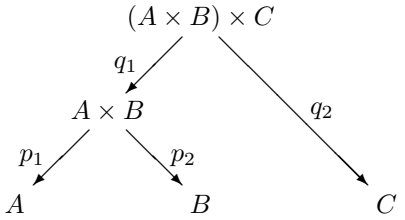
The following uniqueness theorem for general finite products can be proved in the same way as Theorem 6.2.2.

6.3.2 Theorem *Suppose $A_1, A_2, \ldots, A_n$ are objects of a category $\mathscr{C}$ and that A, with projections $p_i : A \longrightarrow A_i$, and B, with projections $q_i : B \longrightarrow A_i$, are products of these objects. Then there is a unique arrow $\phi : A \longrightarrow B$ for which $q_i \circ \phi = p_i$ for $i = 1, \ldots, n$. Moreover, ϕ is an isomorphism.*

Propositions 6.2.3, 6.2.11 and 6.2.18 also generalize in the obvious way to n-ary products.

6.3.3 Binary products give ternary products An important consequence of the definition of ternary product is that in any category with binary products, and any objects A, B and C, either of $(A \times B) \times C$ and $A \times (B \times C)$ can be taken as ternary products $A \times B \times C$ with appropriate choice of projections.

We prove this for $(A \times B) \times C$. Writing p_i, $i = 1, 2$, for the projections which make $A \times B$ a product of A and B and q_i, $i = 1, 2$ for the projections which make $(A \times B) \times C$ a product of $A \times B$ and C, we claim that



is a product diagram with vertex $(A \times B) \times C$ and projections $p_1 \circ q_1 : (A \times B) \times C \longrightarrow A$, $p_2 \circ q_1 : (A \times B) \times C \longrightarrow B$, and $q_2 : (A \times B) \times C \longrightarrow C$.

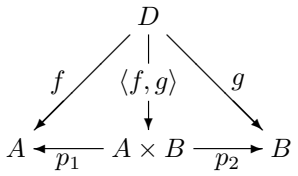
Suppose that $f : D \longrightarrow A$, $g : D \longrightarrow B$, and $h : D \longrightarrow C$ are given. We must construct an arrow $u : D \longrightarrow (A \times B) \times C$ with the property that

(a) $p_1 \circ q_1 \circ u = f$,

(b) $p_2 \circ q_1 \circ u = g$, and

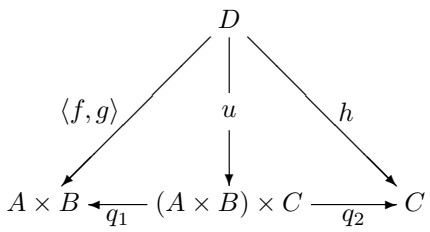
(c) $q_2 \circ u = h$.

Recall that $\langle f, g\rangle$ is the unique arrow making



(6.21)

commute. This induces a unique arrow $u = \langle\langle f, g\rangle, h\rangle$ making



(6.22)

commute.

The fact that (a) through (c) hold can be read directly off these diagrams. For example, for (a), $p_1 \circ q_1 \circ u = p_1 \circ \langle f, g\rangle = f$.

Finally, if u' were another arrow making (a) through (c) hold, then we would have $p_1 \circ q_1 \circ u' = f$ and $p_2 \circ q_1 \circ u' = g$, so by uniqueness of v as defined by (6.21), $v = q_1 \circ u'$. Since $q_2 \circ u'$ must be h, the uniqueness of u in (6.22) means that $u' = u$.

A generalization of this is stated in Proposition 6.3.10 below.

6.3.4 It follows from the discussion in 6.3.3 that the two objects $(A \times B) \times C$ and $A \times (B \times C)$ are pairwise canonically isomorphic (to each other and to any other realization of the ternary product) in a way that preserves the ternary product structure.

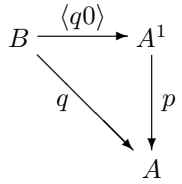
> In elementary mathematics texts the point is often made that 'cartesian product is not associative'. When you saw this you may have thought in your heart of hearts that $(A \times B) \times C$ and $A \times (B \times C)$ are nevertheless really the same. Well, now you know that they are really the same in a very strong sense: they satisfy the *same specification* and so carry exactly the same information. The only difference is in *implementation*.

6.3.5 If all the factors in an n-ary product are the same object A, the n-ary product $A \times A \times \cdots \times A$ is denoted A^n. This suggests the possibility of defining the **nullary product** A^0 and the **unary product** A^1.

6.3.6 For nullary products, the definition is: given no objects of the category $\mathscr{C}$, there should be an object we will temporarily call T, with no arrows from it, such that for any other object B and no arrows from B, there is a unique arrow from B to T subject to no commutativity condition.

When the language is sorted out, we see that a nullary product in $\mathscr{C}$ is simply an object T with the property that every other object of the category has exactly one arrow to T. That is, T must be a terminal object of the category, normally denoted 1. Thus, for any object A of the category, we take $A^0 = 1$. (Compare 5.1.6.)

6.3.7 A unary product A^1 of a single object A should have an arrow $p : A^1 \longrightarrow A$ with the property that given any object B and arrow $q : B \longrightarrow A$ there is a unique arrow $\langle q\rangle : B \longrightarrow A^1$ for which



(6.23)

commutes. The identity $\mathrm{id} : A \longrightarrow A$ satisfies this specification for p; given the arrow $q : B \longrightarrow A$, we let $\langle q\rangle = q : B \longrightarrow A$. The fact that $\mathrm{id} \circ \langle q\rangle = q$ is evident, as is the uniqueness of $\langle q\rangle$. It follows that A^1 can always be taken to be A itself, with the identity arrow as the coordinate arrow.

It is straightforward to show that in general an object B is a unary product of A with coordinate $p : B \longrightarrow A$ if and only if p is an isomorphism.

There is therefore a conceptual distinction between A^1 and A. In the category of sets, A^n is often taken to be the set of strings of elements of A of length exactly n. As you may know, in some computer languages, a string of characters of length one is not the same as a single character, mirroring the conceptual distinction made in category theory.

6.3.8 Definition A category has **binary products** if the product of any two objects exists. It has **canonical binary products** if a specific product diagram is given for each pair of objects. Thus a category with canonical binary products is a category with extra structure given for it. Precisely, a canonical binary products structure on a category $\mathscr{C}$ is a function from $\mathscr{C}_0 \times \mathscr{C}_0$ to the collection of product diagrams in $\mathscr{C}$ which takes a pair (A, B) to a diagram of the form (6.2).

The fact that A and id_A can be taken as a product diagram for any object A of any category means that every category can be given a canonical unary product structure. This is why the distinction between A and A^1 can be and often is ignored.

6.3.9 Definition A category has **finite products** or is a **cartesian category** if the product of any finite number of objects exists. This includes nullary products – in particular, a category with finite products has a terminal object. The category has **canonical finite products** if every finite list of objects has a specific given product.

The following proposition is proved using constructions generalizing those of 6.3.3.

6.3.10 Proposition *If a category has a terminal object and binary products, then it has finite products.*

6.3.11 Example Let $\mathbf{N}$ be the set of nonnegative integers with the usual ordering. It is easy to see that the category determined by $(\mathbf{N}, \leq)$ has all binary products but no terminal object.

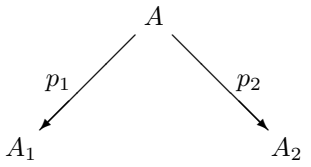
6.3.12 **Set**, **Grf** and **Cat** all have finite products. In fact, a choice of definition for ordered pairs in **Set** provides canonical products not only for **Set** but also for **Grf** and **Cat**, since products in those categories are built using cartesian products of sets.

6.3.13 Record types To allow operations depending on several variables in a functional programming language L (as discussed in 2.2.1), it is reasonable to assume that for any types A and B the language has a record type P and two field selectors $P.A : P \longrightarrow A$ and $P.B : P \longrightarrow B$. If we insist that the data in P be determined completely by those two fields, it follows that for any pair of operations $f : X \longrightarrow A$ and $g : X \longrightarrow B$ there ought to be a unique operation $\langle f, g\rangle : X \longrightarrow P$ with the property that $P.A \circ \langle f, g\rangle = f$ and $P.B \circ \langle f, g\rangle = g$. This would make P the product of A and B with the selectors as product projections.

For example, a record type `PERSON` with fields `NAME` and `AGE` could be represented as a product cone whose base diagram is defined on the discrete graph with two nodes `NAME` and `AGE`. If `HUMAN` is a variable of type `PERSON`, then the field selector `HUMAN.AGE` implements the coordinate projection indexed by `AGE`. This example is closer to the spirit of category theory than the cone in Diagram (6.2); there, the index graph has nodes 1 and 2, which suggests an ordering of the nodes (and the projections) which is in fact spurious.

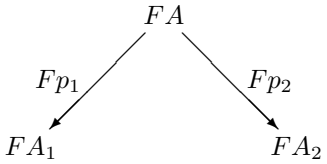
Thus to say that one can always construct record types in a functional programming language L is to say that the corresponding category $C(L)$ has finite products. (See Poigné, [1986].)

6.3.14 Functors that preserve products Let $F : \mathscr{A} \longrightarrow \mathscr{B}$ be a functor between categories. Suppose that



(6.24)

is a (binary) product diagram in $\mathscr{A}$. We say that F **preserves the product** if



is a product diagram in $\mathscr{B}$. It is important to note that F must preserve the diagram, not merely the object A. It is possible for F to take A to an object which is isomorphic to a product, but do the wrong thing on projections.

F **preserves canonical products** if $\mathscr{A}$ and $\mathscr{B}$ have canonical products and F preserves the canonical product diagrams.

Similar definitions can be made about a product diagram of any family of objects. A functor is said to **preserve finite products**, respectively **all products** if it preserves every finite product diagram, respectively all product diagrams. Preserving canonical product diagrams is defined analogously.

We have a proposition related to Proposition 6.3.10.

6.3.15 Proposition *If a functor preserves terminal objects and binary products, it preserves all finite products.*

Two important examples of this is given by Propositions 6.3.16 and 6.3.17 below.

6.3.16 Proposition *Any covariant hom functor preserves products.*

Proof. It is easy to prove that a covariant hom functor preserves terminal objects. Now suppose

$$\begin{array}{ccc} & A \times B & \\ {\scriptstyle p_1}\swarrow & & \searrow{\scriptstyle p_2} \\ A & & B \end{array} \tag{6.25}$$

is a product diagram. We must show that for any object C,

$$\begin{array}{ccc} & \mathrm{Hom}(C, A \times B) & \\ {\scriptstyle \mathrm{Hom}(C,p_1)}\swarrow & & \searrow{\scriptstyle \mathrm{Hom}(C,p_2)} \\ \mathrm{Hom}(C, A) & & \mathrm{Hom}(C, B) \end{array} \tag{6.26}$$

is a product diagram in **Set**. In 6.2.7, we defined the function

$$\pi C : \mathrm{Hom}(C, A) \times \mathrm{Hom}(C, B) \longrightarrow \mathrm{Hom}(C, A \times B)$$

which is a bijection that takes (f, g) to $\langle f, g\rangle$. Let $q_1 : \mathrm{Hom}(C, A) \times \mathrm{Hom}(C, B) \longrightarrow \mathrm{Hom}(C, A)$ be the first coordinate function $(f, g) \mapsto f$ in **Set**, and similarly for q_2. Then

$$\mathrm{Hom}(C, p_1)\langle f, g\rangle = p_1 \circ \langle f, g\rangle = f = q_1(f, g)$$

and similarly $\mathrm{Hom}(C, p_2)\langle f, g\rangle = q_2(f, g)$, so $p_i \circ \pi C = q_i$ for $i = 1, 2$. Hence Diagram (6.26) is a product diagram in **Set** by Proposition 6.2.3. □

If F is a functor that preserves products, then any functor isomorphic to it preserves products (the proof is an instructive diagram chase), so that it follows from Proposition 6.3.16 that any representable functor preserves products.

6.3.17 Proposition *The second Yoneda embedding*

$$J : \mathscr{C} \longrightarrow \mathbf{Func}(\mathscr{C}^{\mathrm{op}}, \mathbf{Set})$$

(see 5.7.8) preserves products.

Proof. By definition, $J(C) = \mathrm{Hom}(-, C)$ for an object C of $\mathscr{C}$, and if $f : C \longrightarrow D$, $J(f) = \mathrm{Hom}(-, f) : \mathrm{Hom}(-, C) \longrightarrow \mathrm{Hom}(-, D)$. For each X in $\mathscr{C}$,

$$\begin{array}{ccc} & \mathrm{Hom}(X, A) \times \mathrm{Hom}(X, B) & \\ {\scriptstyle p_1}\swarrow & & \searrow{\scriptstyle p_2} \\ \mathrm{Hom}(X, A) & & \mathrm{Hom}(X, B) \end{array}$$

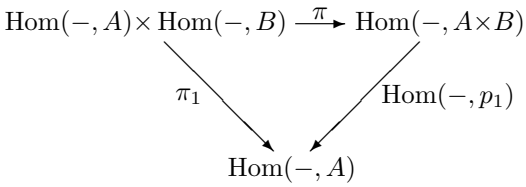
(where p_1 and p_2 are the ordinary projections in **Set**) is a product cone in **Set**. Let

$$\pi_1 : \mathrm{Hom}(-, A) \times \mathrm{Hom}(-, B) \longrightarrow \mathrm{Hom}(-, A)$$

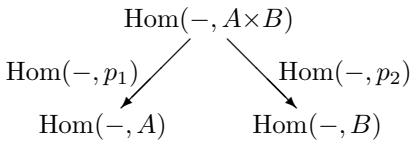
be the natural transformation whose component at X is p_1, and similarly define π_2. Then by the proof of Proposition 6.2.20,

$$\begin{array}{ccc} & \mathrm{Hom}(-, A) \times \mathrm{Hom}(-, B) & \\ {\scriptstyle \pi_1}\swarrow & & \searrow{\scriptstyle \pi_2} \\ \mathrm{Hom}(-, A) & & \mathrm{Hom}(-, B) \end{array}$$

is a product cone in $\mathbf{Func}(\mathscr{C}^{\mathrm{op}}, \mathbf{Set})$. It is easy to show that the diagram



commutes, where π is the natural transformation defined in Proposition 6.2.14. A similar diagram for π_2 also commutes. It now follows from Proposition 6.2.3 that

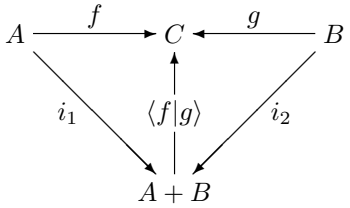


is a product diagram in $\mathbf{Func}(\mathscr{C}^{\mathrm{op}}, \mathbf{Set})$, which is what is required to show that J preserves binary products. It is easy to show that it preserves terminal objects. □

6.4 Sums

A sum in a category is a product in the dual category. This definition spells that out:

6.4.1 Definition The **sum**, also called the **coproduct**, $A + B$ of two objects in a category consists of an object called $A + B$ together with arrows $i_1 : A \longrightarrow A + B$ and $i_2 : B \longrightarrow A + B$ such that given any arrows $f : A \longrightarrow C$ and $g : B \longrightarrow C$, there is a unique arrow $\langle f|g\rangle : A + B \longrightarrow C$ for which

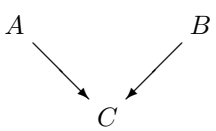


commutes.

The arrows i_1 and i_2 are called the **canonical injections** or the **inclusions** even in categories other than **Set**. These arrows need not be monomorphisms.

6.4.2 More generally, one can define the sum of any finite or infinite indexed set of objects in a category. The sum of a family $A_1, \ldots, A_n$ is denoted $\sum_{i=1}^{n} A_n$ or $\coprod_{i=1}^{n} A_n$ (the latter symbol is the one typically used by those who call the sum the 'coproduct'). Theorems such as Theorem 6.2.2 and Propositions 6.2.3, 6.2.11 and 6.3.2 are stateable in the opposite category and give uniqueness theorems for sums (this depends on the fact that isomorphisms are isomorphisms in the opposite category). The functor represented by the sum of A and B is $\mathrm{Hom}(A, -) \times \mathrm{Hom}(B, -)$ and the universal element is the pair of canonical injections. (Compare Proposition 6.2.14 and the discussion which follows.)

6.4.3 Definition A **binary discrete cocone** in a category $\mathscr{C}$ is a diagram of the form



It is a **sum cocone** if the two arrows shown make C a sum of A and B.

6.4.4 The notions of a category having sums or of having canonical sums, and that of a functor preserving sums, are defined in the same way as for products. The notion corresponding to $f \times g$ for two arrows f and g is denoted $f + g$.

6.4.5 Sums in Set In the category of sets, one can find a sum of two sets in the following way. If S and T are sets, first consider the case that S and T are disjoint. In that case the set $S \cup T$, together with the inclusion functions $S \longrightarrow S \cup T \longleftarrow T$ is a sum cocone. Given $f : S \longrightarrow C$ and $g : T \longrightarrow C$, then $\langle f|g\rangle(s) = f(s)$ for $s \in S$ and $\langle f|g\rangle(t) = g(t)$ for $t \in T$. In this case, the graph of $\langle f|g\rangle$ is the union of the graphs of f and g.

In the general case, all we have to do is find sets S' and T' isomorphic to S and T, respectively, that are disjoint. The union of those two sets is a sum of S' and T' which are the same, as far as mapping properties, as S and T. The usual way this is done is as follows: let

$$S' = S_0 = \{(s, 0) \mid s \in S\} \quad \text{and} \quad T' = T_1 = \{(t, 1) \mid t \in T\}$$

These sets are disjoint since the first is a set of ordered pairs each of whose second entries is a 0, while the second is a set of ordered pairs each of whose second entries is a 1. The arrow $i_1 : S \longrightarrow S' \cup T'$ takes s to $(s, 0)$, and i_2 takes t to $(t, 1)$. If $f : S \longrightarrow C$ and $g : T \longrightarrow C$, then $\langle f|g\rangle(s, 0) = f(s)$ and $\langle f|g\rangle(t, 1) = g(t)$.

> Note that it will not do to write $S_0 = S \times \{0\}$ and $T_1 = T \times \{1\}$ since our specification of products does not force us to use ordered pairs and, in fact, S is itself a possible product of S with either $\{0\}$ or $\{1\}$.

Our notation $S + T$ for the disjoint union of two sets in **Set** conflicts with a common usage in which S and T are sets of numbers and $S + T$ denotes the set of all their sums. For this reason, many use the notation $S \amalg T$ for what we call $S + T$. We will use the notation $S + T$ here but will remind you by referring to it as the disjoint union.

6.4.6 Sums and products in posets The **least upper bound** or **supremum** of two elements x and y in a poset is an element z with the property that $x \leq z$, $y \leq z$ (z is thus an upper bound) and if for some element w, $x \leq w$ and $y \leq w$, then necessarily $z \leq w$ (z is the – necessarily unique – least upper bound). In the category corresponding to the poset, the supremum of two elements is their categorical sum. We have already seen the dual idea of greatest lower bound or infimum in 6.1.9.

A poset whose corresponding category has all finite sums is called a **sup semilattice** or an **upper semilattice**. The sup of s and t is generally denoted $s \vee t$ and the minimum element (initial object) is denoted 0. In this situation, the minimum element is often called 'bottom'. A poset with all finite products is similarly an **inf semilattice** or **lower semilattice**. A **homomorphism** of sup semilattices is a function such that $f(s \vee t) = f(s) \vee f(t)$. Such a function is monotone because if $s \leq t$ then $t = s \vee t$.

A poset with both finite sups and finite infs is a **lattice**. Some authors assume only binary sups and infs so that a lattice need not have a maximum or minimum element. A good source for the theory of lattices is [Davey and Priestley, 1990].

6.4.7 Sums and products in Rel and Pfn Let S and T be two sets. Their sum in **Rel**, the category of sets and relations (2.1.14) and in **Pfn**, the category of sets and partial functions (2.1.13) is the same as in **Set**: the disjoint union $S + T$. For **Pfn**, the canonical injections are the same as for **Set**, and for **Rel** they are the graphs of the canonical injections for **Set**.

We will verify this for **Rel** (primarily to show the subtleties involved) and leave the claim for **Pfn** as an exercise. Set $i_1 = \{(s, (s, 0)) \mid s \in S\}$ and $i_2 = \{(t, (t, 1)) \mid t \in T\}$. For relations $\alpha : S \longrightarrow X$ and $\beta : T \longrightarrow X$, define

$$\langle \alpha|\beta\rangle = \{((s, 0), x) \mid \text{for all } (s, x) \in \alpha\} \cup \{((t, 1), x) \mid \text{for all } (t, x) \in \beta\}$$

Then

$$\begin{aligned}
\langle \alpha|\beta\rangle \circ i_1 &= \{(r, x) \mid \exists z \text{ such that } (r, z) \in i_1 \text{ and } (z, x) \in \langle \alpha|\beta\rangle\} \\
&= \{(r, x) \mid r \in S \text{ and } (r, (r, 0)) \in i_1 \text{ and } ((r, 0), x) \in \langle \alpha|\beta\rangle\} \\
&= \{(r, x) \mid r \in S \text{ and } ((r, 0), x) \in \langle \alpha|\beta\rangle\} \\
&= \{(r, x) \mid r \in S \text{ and } (r, x) \in \alpha\} \\
&= \alpha
\end{aligned}$$

and similarly for i_2 and β.

Now suppose we omitted a particular element $((s_0, 0), x_0)$ from $\langle\alpha|\beta\rangle$. Since i_1 has the functional property, the only z for which $(s_0, z) \in i_1$ is $(s_0, 0)$, but then $((s_0, 0), x_0) \notin \langle\alpha|\beta\rangle$. On the other hand, if we insert an extra element (r, x) into $\langle\alpha|\beta\rangle$, suppose without loss of generality that $r = (s_0, 0)$ for $s \in S$ and $(s, x) \notin \alpha$. Then $(s, (s_0, 0)) \in i_1$ and $((s_0, 0), x) \in \langle\alpha|\beta\rangle$ but $(s, x) \notin \alpha$. So $\langle\alpha, \beta\rangle$ is the only relation from the disjoint union $S + T$ to X such that $\langle\alpha, \beta\rangle \circ i_1 = \alpha$ and $\langle\alpha, \beta\rangle \circ i_2 = \beta$.

The product of two sets S and T in **Rel** is also the disjoint union $S + T$. It must be that because that is the sum and **Rel** is isomorphic to $\mathbf{Rel}^{\mathrm{op}}$. It follows that $p_1 = \{((s, 0), s) \mid s \in S\}$ (the opposite of the graph of i_1) and similarly for p_2.

The product of S and T in **Pfn** is quite different: it is the disjoint union $S \times T + S + T$. The projection p_1 is defined by

$$p_1(r) = \begin{cases} s & \text{if } r = (s, t) \in S \times T \\ r & \text{if } r \in S \\ \text{undefined} & \text{otherwise} \end{cases}$$

6.4.8 Finite sums in programming languages In 6.3.13, we have described products in the category corresponding to a programming language as records. Sums play a somewhat subtler role.

If A and B are types, the sum $A + B$ can be thought of as the free variant or free union of the types A and B. If we are to take this seriously, we have to consider the canonical structure maps $i_1 : A \longrightarrow A + B$ and $i_2 : B \longrightarrow A + B$. These are type conversions; i_1 converts something of type A to something of the union type. Such type conversions are not explicit in languages such as Pascal or C, because in those languages, the free union is implemented in such a way that the type conversion $i_1 : A \longrightarrow A + B$ can always be described as 'use the same internal representation that it has when it is type A'. Thus in those languages, the effect of an operation with domain '$A + B$' is implementation-dependent.

> Thus sums and products provide some elementary constructions for functional programming languages. Still missing are some constructions such as `IF...THEN...ELSE` and `WHILE` loops or recursion which give the language the full power of a Turing machine. `IF...THEN...ELSE` is discussed in Sections 5.7 and 9.6 of [Barr and Wells, 1999]. Some approaches to recursion are discussed in Chapters 6.6 and 14.2 of [Barr and Wells, 1999]. Wagner [1986a, 1986b] discusses these constructions and others in the context of more traditional imperative programming languages.

6.5 Deduction systems as categories

In this section, we describe briefly the connection between formal logical systems and categories.

6.5.1 A **deduction system** has **formulas** and **proofs**. The informal idea is that the formulas are statements, such as $x \leq 7$, and the proofs are valid lines of reasoning starting with one formula and ending with another; for example, it is valid in high school algebra (which is a deductive system) to prove that if $x \leq 7$ then $2x \leq 14$. We will write a proof p which assumes A and deduces B as $p : A \longrightarrow B$.

Typically, *formal* deduction systems define the formulas by some sort of context free grammar, a typical rule of which might be: 'If A and B are formulas, then so is $A \wedge B$'. The valid proofs are composed of chains of applications of certain specified **rules of inference**, an example of which might be: 'From $A \wedge B$ it is valid to infer A'.

6.5.2 Assumptions on a deduction system As in the case of functional programming languages (see 2.2.4), certain simple assumptions on a deduction system, however it is defined, produce a category.

DS–1 For any formula A there is a proof $\mathrm{id}_A : A \longrightarrow A$.

DS–2 Proofs can be composed: if you have proofs $p : A \longrightarrow B$ and $q : B \longrightarrow C$ then there is a valid proof $p; q : A \longrightarrow C$.

DS–3 If $p : A \longrightarrow B$ is a proof then $p; \mathrm{id}_B$ and $\mathrm{id}_A; p$ are both the *same* proof as p.

DS–4 If p, q and r are proofs, then $(p;q);r$ must be the same proof as $p;(q;r)$, which could be denoted $p;q;r$.

These requirements clearly make a deduction system a category.

One could take a deduction system and impose the minimum requirements just given to produce a category. One could on the other hand go all the way and make any two proofs A to B the same. In that case, each arrow in the category stands for the usual notion of deducible. The choice of an intermediate system of identification could conceivably involve delicate considerations.

6.5.3 Definition A **conjunction calculus** is a deduction system with a formula true and a formula $A \wedge B$ for any formulas A and B, which satisfies CC–1 through CC–5 below for all A and B.

CC–1 There is a proof $A \longrightarrow \text{true}$.

CC–2 If $u : A \longrightarrow \text{true}$ and $v : A \longrightarrow \text{true}$ are proofs, then $u = v$.

CC–3 There are proofs $p_1 : A \wedge B \longrightarrow A$ and $p_2 : A \wedge B \longrightarrow B$ with the property that given any proofs $q_1 : X \longrightarrow A$ and $q_2 : X \longrightarrow B$ there is a proof $\langle q_1, q_2 \rangle : X \longrightarrow A \wedge B$.

CC–4 For any proofs $q_1 : X \longrightarrow A$ and $q_2 : X \longrightarrow B$, $p_1 \circ \langle q_1, q_2 \rangle = q_1$ and $p_2 \circ \langle q_1, q_2 \rangle = q_2$.

CC–5 For any proof $h : Y \longrightarrow A \wedge B$, $\langle p_1 \circ h, p_2 \circ h \rangle = h$.

Similar constructions may be made using sums to get a disjunction calculus. Cartesian closed categories (the subject of Chapter 7) give an implication operator and quantifiers are supplied in a topos (Chapter 11). These topics are pursued in detail in [Lambek and Scott, 1986], [Makkai and Reyes, 1977] and [Bell, 1988].

Deductive systems typically involve a distributive law between "and" and "or". A **distributive category** is a category with products and sums that distribute over each other. Expositions of distributive categories with applications to computing science may be found in in [Walters, 1991], [Cockett, 1993], and [Barr and Wells, 1999], Section 5.7. See also [Walters, 1992].

7. Cartesian closed categories

A cartesian closed category is a type of category that as a formal system has the same expressive power as a typed λ-calculus. In Sections 7.1 and 7.2, we define cartesian closed categories and give some of their properties. In Section 7.3, we describe the concept of typed λ-calculus. Section 7.4 describes the constructions involved in translating from one formalism to the other, without proof. Sections 7.1 and 7.2 are needed for Chapter 11. Monoidal closed categories, discussed in Chapter 12, are a generalization of cartesian closed categories, but the discussion in Chapter 12 is independent of this chapter. Other than for the chapters just mentioned, this chapter is not needed in the rest of these notes.

Most of the cartesian closed categories considered in computer science satisfy a stronger property, that of being 'locally cartesian closed', meaning that every slice is cartesian closed. This is discussed in Section 13.4 of [Barr and Wells, 1999].

A basic reference for cartesian closed categories and their connection with logic is [Lambek and Scott, 1986]. See also [Huet, 1986] and [Mitchell and Scott, 1989]. The text by Gunter [1992] gives a systematic treatment of programming language semantics in terms of the ideas of this chapter. Substantial applications to computing may be found in [Cousineau, Curien and Mauny, 1985], [Curien, 1986], [Hagino, 1987b] and [Dybkjaer and Melton, 1993]. The construction of fixed points in cartesian closed categories is discussed in [Barr and Wells, 1999], Section 6.6, and in [Barr, 1990], [Huwig and Poigné, 1990] and [Backhouse *et al.*, 1995].

In this chapter and later, we frequently use the name of an object to stand for the identity arrow on the object: thus 'A' means 'id_A'. This is common in the categorical literature because it saves typographical clutter.

7.1 Cartesian closed categories

7.1.1 Functions of two variables If S and T are sets, then an element of $S \times T$ can be viewed interchangeably as a *pair* of elements, one from S and one from T, or as a *single* element of the product. If V is another set, then a function $f : S \times T \to V$ can interchangeably be viewed as a function of a single variable ranging over $S \times T$ or as a function of two variables, one from S and one from T. Conceptually, we must distinguish between these two points of view, but they are equivalent.

In a more general category, the notion of function of two variables should be understood as meaning an arrow whose domain is a product (see 6.2.6). Under certain conditions, such a function can be converted to a function of one variable with values in a 'function object'. We now turn to the study of this phenomenon.

7.1.2 To **curry** a function of two variables is to change it into a function of one variable whose values are functions of one variable. Precisely, let S, T and V be sets and $f : S \times T \to V$ a function. Let $[S \to T]$ denote the set of functions from S to T. Then there is a function $\lambda f : S \to [T \to V]$ defined by letting $\lambda f(s)$ be the function whose value at an element $t \in T$ is $f(s,t)$. The passage from f to λf is called **currying** f. As an example, the definition on arrows of the functor F_α obtained from a monoid action as in 4.2.4 is obtained by currying α.

In the other direction, if $g : S \to [T \to V]$ is a function, it induces a function $f : S \times T \to V$ defined by $f(s,t) = [g(s)](t)$. This construction produces an inverse to λ that determines an isomorphism

$$\mathrm{Hom}_{\mathbf{Set}}(S \times T, V) \cong \mathrm{Hom}_{\mathbf{Set}}(S, [T \to V])$$

These constructions can readily be stated in categorical language. The result is a theory that is equivalent to the typed λ-calculus (in the sense of Section 7.3) and has several advantages over it.

7.1.3 Definition A category $\mathscr{C}$ is called a **cartesian closed category** if it satisfies the following:

CCC–1 There is a terminal object 1.

CCC–2 Each pair of objects A and B of $\mathscr{C}$ has a product $A \times B$ with projections $p_1 : A \times B \to A$ and $p_2 : A \times B \to B$.

CCC–3 For every pair of objects A and B, there is an object $[A \to B]$ and an arrow $\mathrm{eval} : [A \to B] \times A \to B$ with the property that for any arrow $f : C \times A \to B$, there is a unique arrow $\lambda f : C \to [A \to B]$ such that the composite

$$C \times A \xrightarrow{\lambda f \times A} [A \to B] \times A \xrightarrow{\mathrm{eval}} B$$

is f.

7.1.4 Terminology Traditionally, $[A \to B]$ has been denoted B^A and called the **exponential object**, and A is then called the exponent. The exponential notation is motivated by the following special case: if $\mathscr{C}$ is the category of sets and $n = \{0, 1, \ldots, n-1\}$, the standard set with n elements, then B^n is indeed the set of n-tuples of elements of B.

In CCC–3, there is a different arrow eval for each pair of objects A and B. When necessary, we will write the arrow $\mathrm{eval} : [A \to B] \times A \to B$ as eval^A_B. This collection of arrows for a fixed A forms the counit of an adjunction, as defined in 9.2.5. Because of that, λf is often called the **adjoint transpose** of f.

7.1.5 In view of Proposition 6.3.10, CCC–1 and CCC–2 could be replaced by the requirement that $\mathscr{C}$ have finite products. There is a slight notational problem in connection with this. We have used p_1 and p_2 for the two projections of a binary product. We will now use

$$p_j : \prod A_i \to A_j$$

for $j = 1, \ldots, n$ to denote the jth projection from the n-ary product. This usage should not conflict with the previous usage since it will always be clear what the domain is.

A related problem is this. Condition CCC–3 appears to treat the two factors of $C \times A$ asymmetrically, which is misleading since of course $C \times A \cong A \times C$. Products are of indexed sets of objects (see 6.1.3) not necessarily indexed by an ordered set, even though our notation appears to suggest otherwise. It gets even worse with n-ary products, so we spell out the notation we use more precisely.

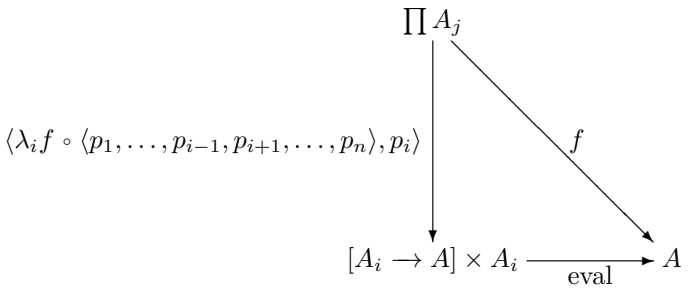
7.1.6 Proposition *In any cartesian closed category, for any objects A_1,...,A_n and A and any $i =$ 1,...,n, there is an object $[A_i \to A]$ and an arrow*

$$\mathrm{eval} : [A_i \to A] \times A_i \to A$$

such that for any $f : \prod A_j \to A$, there is a unique arrow

$$\lambda_i f : \prod_{j \neq i} A_j \to [A_i \to A]$$

such that the following commutes:



In **Set**, $\lambda_i f$ gives a function from A_i to A for each $(n-1)$-tuple

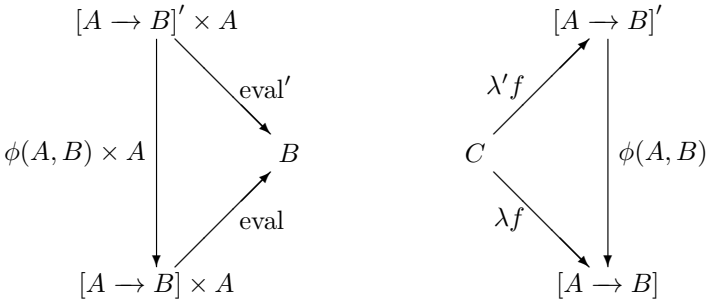
$$(a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$$

7.1.7 Evaluation as universal element For fixed objects A and B of a cartesian closed category, let $F_{A,B}$ denote the functor $\mathrm{Hom}(- \times A, B)$, so that if $g : D \to C$, $F_{A,B}(g) : \mathrm{Hom}(C \times A, B) \to \mathrm{Hom}(D \times A, B)$ takes $f : C \times A \to B$ to $f \circ (g \times A)$. CCC–3 says that $\mathrm{eval} : [A \to B] \times A \to B$ is a universal element for $F_{A,B}$. This is true for every object A and B and by Proposition 5.7.16 implies that the maps $f \mapsto \lambda f$ form a natural isomorphism of functors:

$$\mathrm{Hom}(- \times A, B) \cong \mathrm{Hom}(-, [A \to B]) \tag{7.1}$$

It follows that a category with finite products can be a cartesian closed category in essentially only one way. The following proposition makes this precise. It is a special case of the fact that adjoints are essentially unique (see 9.3.4).

7.1.8 Proposition *Let $\mathscr{C}$ be a category with finite products. Suppose that for every pair of objects A and B, there are objects $[A \to B]$ and $[A \to B]'$ and arrows* $\mathrm{eval} : [A \to B] \times A \to B$ *and* $\mathrm{eval}' : [A \to B]' \times A \to B$. *Suppose these have the property that for any arrow $f : C \times A \to B$, there are unique arrows $\lambda f : C \to [A \to B]$, $\lambda' f : C \to [A \to B]'$ for which* $\mathrm{eval} \circ (\lambda f \times A) = \mathrm{eval}' \circ (\lambda' f \times A) = f$. *Then for all objects A and B, there is a unique arrow $\phi(A, B) : [A \to B]' \to [A \to B]$ such that for every arrow $f : C \times A \to B$ the following diagrams commute:*



Moreover, ϕ is an isomorphism.

7.1.9 Example The first example of a cartesian closed category is the category of sets. For any sets A and B, $[A \to B]$ is the set of functions from A to B, and eval_B^A is the evaluation or apply function. The meaning of λf is discussed above in 7.1.2. Note that in the case of **Set**, $[A \to B]$ *is* $\mathrm{Hom}_{\mathbf{Set}}(A, B)$.

7.1.10 Example A Boolean algebra B is a poset, so corresponds to a category $C(B)$. This category is a cartesian closed category. The terminal object of a Boolean algebra B is 1, so $C(B)$ satisfies CCC–1. Since B has the infimum of any two elements, $C(B)$ has binary products (see 6.1.9) and so satisfies CCC–2.

To prove CCC–3, define $[a \to b]$ to be $\neg a \vee b$. To show that eval_b^a exists requires showing that $[a \to b] \wedge a \leq b$. This can be seen from the following calculation, which uses the distributive law:

$$\begin{aligned} [a \to b] \wedge a &= (\neg a \vee b) \wedge a \\ &= (\neg a \wedge a) \vee (b \wedge a) \\ &= 0 \vee (b \wedge a) = b \wedge a \leq b \end{aligned}$$

The existence of the λ function requires showing that if $c \wedge a \leq b$ then $c \leq [a \to b]$ (the uniqueness of λf and the fact that $\mathrm{eval} \circ \lambda f \times A = f$ are automatic since no hom set in the category determined by a poset has more than one element). This follows from this calculation, assuming $c \wedge a \leq b$:

$$\begin{aligned} c &= c \wedge 1 = c \wedge (a \vee \neg a) \\ &= (c \wedge a) \vee (c \wedge \neg a) \\ &\leq b \vee (c \wedge \neg a) \leq b \vee \neg a = [a \to b] \end{aligned}$$

7.1.11 Example A poset with all finite (including empty) infs and sups that is cartesian closed as a category is called a **Heyting algebra**. A Heyting algebra is thus a generalization of a Boolean algebra. Heyting algebras correspond to intuitionistic logic in the way that Boolean algebras correspond to classical logic. A Heyting algebra is **complete** if it has sups of all subsets. We use this concept in Chapter 11. A thorough exposition of the elementary properties of Heyting algebras may be found in [Mac Lane and Moerdijk, 1992], section I.8.

7.1.12 Example The category whose objects are graphs and arrows are homomorphisms of graphs is also cartesian closed. If $\mathscr{G}$ and $\mathscr{H}$ are graphs, then the exponential $[\mathscr{G} \to \mathscr{H}]$ (which must be a graph) can be described as follows. Let No denote the graph consisting of a single node and no arrows and Ar the graph with one arrow and two distinct nodes, the nodes being the source and the target of the arrow.

There are two embeddings of No into Ar, which we will call $s, t : \mathrm{No} \to \mathrm{Ar}$, that take the single node to the source and target of the arrow of Ar, respectively. Then $[\mathscr{G} \to \mathscr{H}]$ is the graph whose set of nodes is the set $\mathrm{Hom}(\mathscr{G} \times \mathrm{No}, \mathscr{H})$ of graph homomorphisms from $\mathscr{G} \times \mathrm{No}$ to $\mathscr{H}$ and whose set of arrows is the set $\mathrm{Hom}(\mathscr{G} \times \mathrm{Ar}, \mathscr{H})$ with the source and target functions given by $\mathrm{Hom}(\mathscr{G} \times s, \mathscr{H})$ and $\mathrm{Hom}(\mathscr{G} \times t, \mathscr{H})$, respectively.

Note that in the case of graphs, the object $[\mathscr{G} \to \mathscr{H}]$ is *not* the set $\mathrm{Hom}(\mathscr{G}, \mathscr{H})$ with the structure of a graph imposed on it in some way. Since $[\mathscr{G} \to \mathscr{H}]$ is an object of the category, it must be a graph; but neither its set of objects nor its set of arrows is $\mathrm{Hom}(\mathscr{G}, \mathscr{H})$. In particular, to prove that a category of sets with structure is not cartesian closed, it is not enough to prove that an attempt to put a structure on the hom set must fail.

7.1.13 Example If $\mathscr{C}$ is a small category then $\mathbf{Func}(\mathscr{C}, \mathbf{Set})$ is a category; the objects are functors and the arrows are natural transformations. (See 5.3.1.) The category $\mathbf{Func}(\mathscr{C}, \mathbf{Set})$ is cartesian closed. If $F, G : \mathscr{C} \to \mathbf{Set}$ are two functors, the product $F \times G$ was defined in 6.2.20. The exponential object $[F \to G]$ is the following functor. For an object C, $[F \to G](C)$ is the set of natural transformations from the functor $\mathrm{Hom}(C, -) \times F$ to G. For $f : C \to D$ an arrow of $\mathscr{C}$ and $\alpha : \mathrm{Hom}(C, -) \times F \to G$ a natural transformation, $[F \to G](f)(\alpha)$ has component at an object A that takes a pair (v, x) with $v : D \to A$ and $x \in F(A)$ to the element $\alpha A(v \circ f, x)$ of $G(A)$. See [Mac Lane and Moerdijk, 1992], page 46 for a proof of a more general theorem of which this is a special case.

Example 7.1.12 is a special case of this example. This also implies, for example, that the category of u-structures described in 5.2.5 is cartesian closed, as is the arrow category of **Set** (see 5.2.17).

7.1.14 Example The category **Cat** of small categories and functors is cartesian closed. In this case, we have already given the construction in 5.3.1: for two categories $\mathscr{C}$ and $\mathscr{D}$, $[\mathscr{C} \to \mathscr{D}]$ is the category whose objects are functors from $\mathscr{C}$ to $\mathscr{D}$ and whose arrows are natural transformations between them.

If $F : \mathscr{B} \times \mathscr{C} \to \mathscr{D}$ is a functor, then $\lambda F : \mathscr{B} \to [\mathscr{C} \to \mathscr{D}]$ is the functor defined this way: if B is an object of $\mathscr{B}$, then for an object C of $\mathscr{C}$, $\lambda F(B)(C) = F(B, C)$, and for an arrow $g : C \to C'$, $\lambda F(B)(g)$ is the arrow $F(B, g) : (B, C) \to (B, C')$ of $\mathscr{D}$. For an arrow $f : B \to B'$ of $\mathscr{B}$, $\lambda F(f)$ is the natural transformation from $\lambda F(B)$ to $\lambda F(B')$ with component $\lambda F(f)(C) = F(f, C)$ at an object C of $\mathscr{C}$. It is instructive to check that this does give a natural transformation.

7.1.15 Example (For those familiar with complete partial orders.) The category consisting of ω-CPOs and continuous functions is a cartesian closed category, although the subcategory of strict ω-CPOs and strict continuous maps is not. The category of continuous lattices and continuous functions between them is cartesian closed; a readable proof is in [Scott, 1972], Section 3. Many other categories of domains have been proposed for programming language semantics and many, but not all, are cartesian closed. More about this is in [Scott, 1982], [Smyth, 1983], [Dybjer, 1986] and [Gunter, 1992].

7.1.16 Example In 2.2.2 we described how to regard a functional programming language as a category. If such a language is a cartesian closed category, then for any types A and B, there is a type $[A \to B]$ of functions from A to B. Since that is a type, one can apply programs to data of that type: that is, functions can be operated on by programs. This means that functions are on the same level as other data, often described by saying 'functions are first class objects'.

Proposition 7.1.8 puts strong constraints on making functions into first class objects; if you make certain reasonable requirements on your types $[A \to B]$, there is essentially only one way to do it.

7.1.17 Example When a deduction system (Section 6.5) is a cartesian closed category, the constructions giving the exponential turn out to be familiar rules of logic.

Thus, if A and B are formulas, $[A \to B]$ is a formula; think of it as A implies B. Then eval is a proof allowing the deduction of B from A and $[A \to B]$; in other words, it is *modus ponens*. Given $f : C \times A \to B$, that is, given a proof that C and A together prove B, λf is a proof that deduces $[A \to B]$ from C. This is the rule of detachment.

7.2 Properties of cartesian closed categories

Many nice properties follow from the assumption that a category is cartesian closed. Some of them are easy consequences of theorems concerning adjunctions (Chapter 9). Others can be proved using the Yoneda embedding. We now state some of these basic properties, with some of the proofs outlined.

Given $f : B \to C$ in a cartesian closed category, we have the composite

$$[A \to B] \times A \xrightarrow{\text{eval}_B^A} C \xrightarrow{f} C$$

Given $g : B \to A$ we have

$$[A \to C] \times B \xrightarrow{[A \to C] \times g} [A \to C] \times A \xrightarrow{\text{eval}_C^A} C$$

These are used in the following proposition.

7.2.1 Proposition *Let A be an object of a cartesian closed category $\mathscr{C}$. There are functors $F : \mathscr{C} \to \mathscr{C}$ and $G : \mathscr{C}^{\text{op}} \to \mathscr{C}$ for which*

(i) *$F(B) = [A \to B]$ for an object B, and for an arrow $f : B \to C$,*

$$F(f) = \lambda(f \circ \text{eval}_B^A) : [A \to B] \to [A \to C]$$

(ii) *$G(C) = [A \to C]$ for an object C, and for an arrow $g : B \to A$,*

$$G(g) = \lambda(\text{eval}_C^A \circ [A \to C] \times g) : [A \to C] \to [B \to C]$$

The value of F at $f : B \to C$ is normally written

$$[A \to f] : [A \to B] \to [A \to C]$$

and the value of G at $g : B \to A$ is normally written

$$[g \to C] : [A \to C] \to [B \to C]$$

These functors are called the **internal hom functors** of the cartesian closed category.

Proof. Preservation of the identity is straightforward. We will prove that F preserves composition. The proof for G is similar. We must show that, for $f : B \to C$ and $g : C \to D$, $[A \to f] \circ [A \to g] = [A \to g \circ f]$. By Definition 7.1.3, for any $h : X \to Y$, $[A \to h]$ is the unique arrow such that

$$\begin{array}{ccc} [A \to X] \times A & \xrightarrow{\lambda(h \circ \text{eval}) \times A} & [A \to Y] \times A \\ \Big\downarrow{\scriptstyle \text{eval}} & & \Big\downarrow{\scriptstyle \text{eval}} \\ X & \xrightarrow[h]{} & Y \end{array}$$

commutes. But then both squares in the following diagram commute, so the outer rectangle commutes, so that $[A \to g] \circ [A \to f] = [A \to g \circ f]$.

$$\begin{array}{ccccc} [A \to B]\times A & \xrightarrow{\lambda(f\circ \mathrm{eval})\times A} & [A \to C]\times A & \xrightarrow{\lambda(g\circ \mathrm{eval})\times A} & [A \to D]\times A \\ \downarrow{\scriptstyle \mathrm{eval}} & & \downarrow{\scriptstyle \mathrm{eval}} & & \downarrow{\scriptstyle \mathrm{eval}} \\ B & \xrightarrow[f]{} & C & \xrightarrow[g]{} & D \end{array} \tag{7.2}$$

□

Now that we have Proposition 7.2.1, we can regard both $\mathrm{Hom}(C \times A, B)$ and $\mathrm{Hom}(C, [A \to B])$ as functors of any one of the three variables. Thus for fixed A and B, there are contravariant functors $\mathrm{Hom}(- \times A, B)$ and $\mathrm{Hom}(-, [A \to B])$ (we don't need Proposition 7.2.1 to define these), and we have already seen in 7.1.7 that these functors are naturally isomorphic. Because $C \times A \cong A \times C$, A can be treated in the same way as C. For fixed A and C there are covariant functors $\mathrm{Hom}(C \times A, -)$ and $\mathrm{Hom}(C, [A \to -])$, all defined as composites of functors. The following proposition now follows from Theorem 9.3.5. It is also not difficult to prove directly.

7.2.2 Proposition *The function $f \mapsto \lambda f$ defines a natural isomorphism*

$$\mathrm{Hom}(C \times A, -) \to \mathrm{Hom}(C, [A \to -])$$

The proposition below collects the ways in which $[A \to -]$ behaves like a hom functor.

7.2.3 Proposition *The following isomorphisms hold for any objects A, B and C in a cartesian closed category. The last two isomorphisms hold whenever the requisite initial object or sum exists.*

(i) $[A \to 1] \cong 1$.
(ii) $[1 \to A] \cong A$.
(iii) $[A \times B \to C] \cong [A \to [B \to C]]$.
(iv) $[A \to B] \times [A \to C] \cong [A \to B \times C]$.
(v) $[0 \to A] \cong 1$.
(vi) $[A + B \to C] \cong [A \to C] \times [B \to C]$.

It is instructive to rewrite these isomorphisms using the notation B^A for the object $[A \to B]$.

These isomorphisms are 'natural in all the variables'. This means: fix all but one variable. Then both sides of the isomorphism are functors in the remaining variable in a way analogous to what we did for Proposition 7.2.2, and the isomorphism is a natural isomorphism between those functors. For example, fixing A and C in item (iv), we have naturally isomorphic functors $[A \to -] \times [A \to C]$ and $[A \to (- \times C)]$. Given $f : B \to B'$, $[A \to f]$ is an arrow as defined in Proposition 7.2.1, and so $[A \to f] \times [A \to C]$, meaning $[A \to f] \times \mathrm{id}_{[A \to C]}$, is the product of two arrows as in 6.2.17. $[A \to (- \times C)]$ is similarly defined as a composite of functors.

Each of the isomorphisms in Proposition 7.2.3 can be proved by using fullness of the Yoneda embedding (Theorem 5.7.6). We will prove (iii). For fixed objects B and C we have functors $[- \times B \to C]$ and $[- \to [B \to C]]$ from $\mathscr{C}^{\mathrm{op}}$ to $\mathscr{C}$. Then we have the following chain of natural isomorphisms.

$$\begin{aligned} \mathrm{Hom}(1, [- \times B \to C]) &\cong \mathrm{Hom}(- \times B, C) \\ &\cong \mathrm{Hom}(-, [B \to C]) \\ &\cong \mathrm{Hom}(1, [- \to [B \to C]]) \end{aligned}$$

The second one is that of the isomorphism (7.1), page 75, and the first and third are straightforward. Now Theorem 5.7.6 gives an isomorphism from $[- \times B \to C]$ to $[- \to [B \to C]]$.

7.2.4 Proposition *Let $\mathscr{C}$ be a cartesian closed category with objects A and B. Then*

$$\lambda(\text{eval}) = \text{id}_{[A \longrightarrow B]} : [A \longrightarrow B] \longrightarrow [A \longrightarrow B]$$

Proof. By CCC–3, $\text{eval} \circ ((\lambda(\text{eval})) \times A) = \text{eval} : [A \longrightarrow B] \times A \longrightarrow B$. By the uniqueness requirement of CCC-3, $(\lambda(\text{eval})) \times A = [A \longrightarrow B] \times A$ (the identity arrow) so the first component of $(\lambda(\text{eval}) \times A)$ must be the identity. □

7.3 Typed λ-calculus

In order to describe the connection between cartesian closed categories and typed λ-calculus, we give a brief description of the latter. The material is essentially adapted from [Lambek and Scott, 1984] and [Lambek and Scott, 1986]. It differs from the latter reference in that we do not suppose the existence of a (weak) natural numbers object. The brief discussion in [Lambek, 1986] may be helpful as an overview. The standard reference on the λ-calculus is [Barendregt, 1984], which emphasizes the untyped case.

There is a more general idea of typed λ-calculus in the literature that includes the idea presented here as one extreme. See [Hindley and Seldin, 1986], pp. 168ff.

7.3.1 Definition A **typed λ-calculus** is a formal theory consisting of **types, terms, variables** and **equations**. To each term a, there corresponds a type A, called the type of a. We will write $a \in A$ to indicate that a is a term of type A. These are subject to the following rules:

TL–1 There is a type 1.

TL–2 If A and B are types, then there are types $A \times B$ and $[A \longrightarrow B]$.

TL–3 There is a term $*$ of type 1.

TL–4 For each type A, there is a countable set of terms x_i^A of type A called **variables of type** A.

TL–5 If a and b are terms of type A and B, respectively, there is a term (a, b) of type $A \times B$.

TL–6 If c is a term of type $A \times B$, there are terms $\text{proj}_1(c)$ and $\text{proj}_2(c)$ of type A and B, respectively.

TL–7 If a is a term of type A and f is a term of type $[A \longrightarrow B]$, then there is a term $f\text{`}a$ of type B.

TL–8 If x is a variable of type A and $\phi(x)$ is a term of type B, then $\lambda_{x \in A}\phi(x) \in [A \longrightarrow B]$.

The intended meaning of all the term-forming rules should be clear, with the possible exception of $f\text{`}a$ which is to be interpreted as f applied to a. The notation $\phi(x)$ for the term in TL–8 means that ϕ is a term that *might* contain the variable x. We will use the usual mathematical notation for substitution: if we have called a term $\phi(x)$, then $\phi(a)$ is the term obtained by substituting a for every occurrence of x in the term. In the literature on the λ-calculus, more elaborate notations for substitution are used because they are necessary in complex calculations.

In much of the literature, the symbol ` is omitted; $f\text{`}a$ would be written fa.

7.3.2 Before stating the equations, we need some definitions. We omit, as usual, unnecessary subscripts.

If x is a variable, then x is **free** in the term x. If an occurrence of x is free in either of the terms a or b, then that occurrence of x is free in (a, b). If x occurs freely in either f or a, then that occurrence of x is free in $f\text{`}a$. On the other hand, every occurrence of x is not free (and is called **bound**) in $\lambda_x\phi(x)$.

The term a is **substitutable** for x in $\phi(x)$ if no occurrence of a variable in a becomes bound when every occurrence of x in $\phi(x)$ (if any) is replaced by a. A term is called **closed** if no variable is free in it.

7.3.3 The equations take the form $a =_X a'$, where a and a' are terms of the same type and X is a finite set of variables among which are all variables occurring freely in either a or a'.

TL–9 The relation $=_X$ is reflexive, symmetric and transitive.

TL–10 If a is a term of type 1, then $a =_{\{\}} *$.

TL–11 If $X \subseteq Y$, then $a =_X a'$ implies $a =_Y a'$.

TL–12 $a =_X a'$ implies $f\text{`}a =_X f\text{`}a'$.

TL–13 $f =_X f'$ implies $f`a =_X f'`a$.

TL–14 $\phi(x) =_{X\cup\{x\}} \phi'(x)$ implies $\lambda_x\phi(x) =_X \lambda_x\phi'(x)$.

TL–15 $\mathrm{proj}_1(a,b) =_X a$; $\mathrm{proj}_2(a,b) =_X b$, for $a \in A$ and $b \in B$.

TL–16 $c =_X (\mathrm{proj}_1(c), \mathrm{proj}_2(c))$, for $c \in A \times B$.

TL–17 $\lambda_x\phi(x)`a =_X \phi(a)$, if a is substitutable for x in $\phi(x)$ and $\phi(a)$ is gotten by replacing every free occurrence of x by a in $\phi(x)$.

TL–18 $\lambda_{x\in A}(f`x) =_X f$ provided $x \notin X$ (and is thus not free in f).

TL–19 $\lambda_{x\in A}\phi(x) =_X \lambda_{x'\in A}\phi(x')$ if x' is substitutable for x in $\phi(x)$ and x' is not free in $\phi(x)$ and vice versa.

It is important to understand that the expression $a =_X a'$ does not imply that the terms a and a' are equal. Two terms are equal only if they are identical. The symbol '$=_X$' may be understood as meaning that in any formal interpretation of the calculus, the denotation of the two terms must be the same.

It should be emphasized that these type and term-forming rules and equations are not exhaustive. There may and generally will be additional types, terms and equations. It is simply that a typed λ-calculus must have at least the types, terms and equations described above.

We will usually abbreviate proj_1 and proj_2 by p_1 and p_2.

7.4 λ-calculus to category and back

Both of the concepts of typed λ-calculus and cartesian closed category are adapted to understanding the calculus of functions of several variables, so it is not surprising that they are equivalent. In this section and the next, we describe the constructions which give this equivalence.

7.4.1 Definition of the category Given a typed λ-calculus $\mathscr{L}$, the objects of the category $C(\mathscr{L})$ are the types of $\mathscr{L}$. An arrow from an object A to an object B is an equivalence class of terms of type B with one free variable of type A (which need not actually occur in the terms).

The equivalence relation is the least reflexive, symmetric, transitive relation induced by saying that two such terms $\phi(x)$ and $\psi(y)$ are equivalent if ϕ and ψ are both of the same type, x and y are both of the same type, x is substitutable for y in ψ, and $\phi(x) =_{\{x\}} \psi(x)$, where $\psi(x)$ is obtained from $\psi(y)$ by substituting x for every occurrence of y.

The reason we need equivalence classes is that any two variables of the same type must correspond to the same arrow, the identity, of that object to itself. If $\lambda_{x\in A}x : 1 \to [A \to A]$ is to name the identity arrow of A for any variable $x \in A$, as is intuitively evident, then the arrow corresponding to a variable x of type A must be the identity of A.

The equivalence relation also makes two terms containing a variable of type 1 equivalent (because of TL–10), thus ensuring that 1 will be a terminal object of the category.

7.4.2 Suppose ϕ is a term of type B with at most one free variable x of type A and ψ is a term of type C with at most one free variable y of type B. Note that by replacing, if necessary, x by a variable that is not bound in ψ, we can assume that ϕ is substitutable for y in ψ. We then define the composite of the corresponding arrows to be the arrow which is the equivalence class of the term $\psi(\phi)$ obtained by substituting ϕ for x in ψ.

7.4.3 Proposition *The category $C(\mathscr{L})$ is a cartesian closed category.*

We will not prove this here. The construction we have given follows Lambek and Scott [1986], who give a proof.

We do not need to say what the cartesian closed structure on $C(\mathscr{L})$ is by virtue of Proposition 7.1.8. Nevertheless, the construction is the obvious one. $A \times B$ with proj_1 and proj_2 is the product of A and B, and $[A \to B]$ is the exponential object. If $\phi(x)$ determines an arrow $f : C \times A \to B$, then x must be a variable of type $C \times A$. Using TL–15, we can substitute (z, y) for x in ϕ, getting $\phi(z, y)$ where z is of type C and y is of type A. Then λf is the equivalence class of $\lambda_z\phi(z, y)$.

Note if z and y actually occur in $\phi(z, y)$, then it is not in any equivalence class, since it has two free variables.

7.4.4 Example We will exhibit some calculations that verify two of the properties of a cartesian closed category as an example of how the definition of $C(\mathscr{L})$ works.

Let $\mathscr{L}$ be a typed λ-calculus. Define

$$\Lambda : \mathrm{Hom}_{C(\mathscr{L})}(C \times A \longrightarrow B) \longrightarrow \mathrm{Hom}_{C(\mathscr{L})}(C \longrightarrow [A \longrightarrow B])$$

as follows: for $[\phi(u)] : C \times A \longrightarrow B$ (so that u is a variable of type $C \times A$), $\Lambda([\phi(u)]) = \lambda_x \phi((z, x))$, where z is a variable of type C and x is a variable of type A. Define

$$\Gamma : \mathrm{Hom}_{C(\mathscr{L})}(C \longrightarrow [A \longrightarrow B]) \longrightarrow \mathrm{Hom}_{C(\mathscr{L})}(C \times A \longrightarrow B)$$

as follows: for $[\psi(z)] : C \longrightarrow [A \longrightarrow B]$, $\Gamma([\psi(z)]) = \psi(\mathrm{proj}_1\, u)\text{`}\,\mathrm{proj}_2\, u$, where u is a variable of type $C \times A$. Then Λ and Γ are inverse functions.

We will show one direction of the verification and leave the other as an exercise. The following calculation uses TL–18 and the fact that x does not occur in $\phi(z)$ because by definition of arrow in $C(\mathscr{L})$, $\phi(z)$ contains only one variable and that is not x.

$$\begin{array}{lll} \Lambda(\Gamma(\psi(z))) & = & \Lambda(\psi(\mathrm{proj}_1\, u)\text{`}\,\mathrm{proj}_2\, u) \\ & = & \lambda_x(\psi(\mathrm{proj}_1(z,x))\text{`}\,\mathrm{proj}_2(z,x)) \\ & =_X & \lambda_x \psi(z)\text{`}x \\ & =_X & \psi(z) \end{array}$$

For another example, let A and B be types in a λ-calculus $\mathscr{L}$. Then in $C(\mathscr{L})$, $\mathrm{eval} : [A \longrightarrow B] \times A \longrightarrow B$ can be taken to be the equivalence class of the term $(\mathrm{proj}_1\, u)\text{`}(\mathrm{proj}_2\, u)$, where u is a variable of type $[A \longrightarrow B] \times A$.

To prove this, we must show that for $f : C \times A \longrightarrow B$, $\mathrm{eval} \circ (\lambda f \times A) = f$. First note that $\lambda f \times A = \langle \lambda f \circ p_1, p_2 \rangle : C \times A \longrightarrow [A \longrightarrow B] \times A$. (See 6.2.17.) Now let z be a variable of type C, y a variable of type A which is not in X, and suppose f is determined by a term $\phi(z, y)$ of type B. Then $\lambda f \times A$ is represented by $(\lambda_y \phi(z, y) \circ z, y) =_X (\lambda_y \phi(z, y), y)$ by definition of composition in $C(\mathscr{L})$. Then $\mathrm{eval} \circ (\lambda f \times A)$ is

$$(p_1(\lambda_y \phi(z, y), y))\text{`}p_2(\lambda_y \phi(z, y), y) =_X \lambda_y \phi(z, y)\text{`}y$$

by TL–15 and the easily-proved fact that if $a =_X a'$ and $b =_X b'$ then $(a, b) =_X (a', b')$. Since $y \notin X$, this is $\lambda_y \phi(z, y)$ by TL–18.

7.4.5 Cartesian closed category to λ-calculus Let $\mathscr{C}$ be a cartesian closed category. We will suppose that it has been equipped with given finite products (including, of course, their projections). This means that with each finite indexed set of objects $\{A_i\}$, $i \in I$, there is a given product cone with its projections $p_i : \prod_{i \in I} A_i \longrightarrow A_i$.

The **internal language** of $\mathscr{C}$ is a typed λ-calculus $\mathbf{L}(\mathscr{C})$. We will describe this λ-calculus by following Definition 7.3.1.

The types of $\mathbf{L}(\mathscr{C})$ are the objects of $\mathscr{C}$. The types required by TL–1 and TL–2 are the objects 1, $A \times B$ and $[A \longrightarrow B]$. We will assume there is a countable set of variables x_i^A of type A for each object A as required by TL–4. The terms are defined by TL–3 through TL–8, and equality by TL–9 through TL–19.

7.4.6 Theorem *Let $\mathscr{C}$ be a cartesian closed category with internal language $\mathscr{L}$. Then $C(\mathscr{L})$ is a category equivalent to $\mathscr{C}$.*

Lambek and Scott [1986] prove much more than this. They define what it means for languages to be equivalent and show that if you start with a typed λ-calculus, construct the corresponding category, and then construct the internal language, the language and the original typed λ-calculus are equivalent. They state this in a more powerful way in the language of adjunctions.

8. Limits and colimits

A limit is the categorical version of the concept of an equationally defined subset of a product. For example, a circle of radius 1 is the subset of $\mathbf{R} \times \mathbf{R}$ ($\mathbf{R}$ is the set of real numbers) satisfying the equation $x^2 + y^2 = 1$. Another example is the set of pairs of arrows in a category for which the target of the first is the source of the second: this is the set of pairs for which the composition operation is defined.

A different kind of example is division of one integer by another, which requires that the second argument be nonzero. This can be made an equational condition by building in a Boolean type and a test; the equation then becomes $[y = 0] = \text{false}$. (This can also be handled using finite sums: see Section 5.7 of [Barr and Wells, 1999].) A colimit is similarly the categorical version of a quotient of a sum by an equivalence relation.

This chapter discusses finite limits and colimits in detail, concentrating on certain useful special cases. Infinite limits and colimits are widely used in mathematics, but they are conceptually similar to the finite case, and are not described here.

8.1 Equalizers

If S and T are sets and $f, g : S \to T$ are functions, it is a familiar fact that we can form the subset $Eq(f, g) \subseteq S$ consisting of all elements $s \in S$ for which $f(s) = g(s)$. This concept can be made into a categorical concept by changing it to a specification which turns out to determine a subobject. We will look more closely at the construction in **Set** to see how to make the general categorical construction.

Suppose that $j : Eq(f, g) \to S$ is the inclusion function: $j(u) = u$ for $u \in Eq(f, g)$. Let $h : V \to S$ be a function. Then h factors through $j : Eq(f, g) \to S$ if and only if the image of the function h lies in the subset $Eq(f, g)$ (see 3.3.10). This is the key to the categorical specification of $Eq(f, g)$, Definition 8.1.2 below.

8.1.1 Definition Two arrows $f : A \to B$ and $g : A \to B$ of a category (having the same domain and the same codomain) are a **parallel pair** of arrows. An arrow h with the property that $f \circ h = g \circ h$ is said to **equalize** f and g.

8.1.2 Definition Let $\mathscr{C}$ be a category and $f, g : A \to B$ be a parallel pair of arrows. An **equalizer** of f and g is an object E *together with* an arrow $j : E \to A$ with the following properties:

EQ–1 $f \circ j = g \circ j$.

EQ–2 For each arrow $h : C \to A$ such that $f \circ h = g \circ h$, there is an arrow $k : C \to E$, and only one, such that $j \circ k = h$.

Frequently, E is referred to as an equalizer of f and g without referring to j. Nevertheless, j is a crucial part of the data.

8.1.3 Examples of equalizers In **Set**, an equalizer E of the functions $(x, y) \mapsto x^2 + y^2$ and $(x, y) \mapsto 1$ from $\mathbf{R} \times \mathbf{R}$ to $\mathbf{R}$ is the circle $x^2 + y^2 = 1$. The arrow $j : E \to \mathbf{R} \times \mathbf{R}$ is the inclusion.

Given a graph $\mathscr{G}$, the inclusion of the set of loops in the graph is an equalizer of the source and target functions. This equalizer will be empty if the graph has no loops.

A theorem like Theorem 6.2.2 is true of equalizers as well.

8.1.4 Proposition *If $j : E \to A$ and $j' : E' \to A$ are both equalizers of $f, g : A \to B$, then there is a unique isomorphism $i : E \to E'$ for which $j' \circ i = j$.*

Proof. We give two proofs. The first uses the concept of universal element. Let $\mathscr{C}$ be the category containing the equalizers given. Let $F : \mathscr{C}^{\mathrm{op}} \to \mathbf{Set}$ be the functor for which $F(C) = \{u : C \to A \mid f \circ u = g \circ u\}$, the set of arrows from C which equalize f and g. If $h : D \to C$ let $F(h)(u) = u \circ h$. This makes sense, because if $u \in F(C)$, then $f \circ u \circ h = g \circ u \circ h$, so $u \circ h \in F(D)$. Note that this makes F a

subfunctor of the contravariant hom functor $\mathrm{Hom}(-, A)$. The definition of an equalizer of f and g can be restated this way: an equalizer of f and g is an element $j \in F(E)$ for some object E such that for any $u \in F(C)$ there is a unique arrow $k : C \longrightarrow E$ such that $F(k)(j) = u$. This means j is a universal element of F, so by Corollary 5.7.17, any two equalizers $j \in F(E)$ and $j' \in F(E')$ are isomorphic by a unique arrow $i : E \longrightarrow E'$ such that $F(i)(j') = j$. That is, $j = j' \circ i$, as required.

This method of constructing a functor of which the given limit is a universal element is a standard method in category theory. We have already seen it used in the proof of Proposition 6.2.14 and in the discussion of the uniqueness of eval in 7.1.7.

Here is a direct proof not using universal elements: the fact that $f \circ j' = g \circ j'$ implies the existence of a unique arrow $h : E' \longrightarrow E$ such that $j' = j \circ h$. The fact that $f \circ j = g \circ j$ implies the existence of a unique arrow $h' : E \longrightarrow E'$ such that $j = j' \circ h'$. Then $j \circ h \circ h' = j' \circ h' = j = j \circ \mathrm{id}_E$ and the uniqueness part of the definition of equalizer implies that $h \circ h' = \mathrm{id}_E$. By symmetry, $h' \circ h = \mathrm{id}_{E'}$. □

8.1.5 Proposition *Let $j : E \longrightarrow A$ be an equalizer of the pair of arrows $f, g : A \longrightarrow B$. Then j is a monomorphism. Moreover, any two equalizers of f and g belong to the same subobject of A.*

Proof. To see that j is monic, suppose $h, k : C \longrightarrow E$ with $j \circ h = j \circ k = l$. Then $f \circ l = f \circ j \circ k = g \circ j \circ k$ so there is a unique arrow $m : C \longrightarrow E$ with $j \circ m = l$. But both h and k are such arrows and so $h = k$.

Now suppose j and j' are equalizers of f and g. In the notation of Proposition 8.1.4, i and i^{-1} are the arrows required by the definition of subobject in 3.3.12, since $j' \circ i = j$ and $j \circ i^{-1} = j'$. □

8.1.6 More generally, if $f_1, \ldots, f_n$ are all arrows from A to B, then an object E together with an arrow $j : E \longrightarrow A$ is an equalizer of $f_1, \ldots, f_n$ if it has the property that an arbitrary arrow $h : C \longrightarrow A$ factors uniquely through j if and only if $f_1 \circ h = \cdots = f_n \circ h$. Having equalizers of parallel pairs implies having equalizers of all finite lists,

8.1.7 Regular monomorphisms A monomorphism $e : S \longrightarrow T$ in a category is **regular** if e is an equalizer of a pair of arrows.

8.1.8 Proposition *An arrow in a category that is both an epimorphism and a regular monomorphism is an isomorphism.*

Proof. Let $f : A \longrightarrow B$ be both an epimorphism and an equalizer of $g, h : B \longrightarrow C$. Since $g \circ f = h \circ f$ and f is epi, $g = h$. Then $g \circ \mathrm{id}_B = h \circ \mathrm{id}_B$ so there is a $k : B \longrightarrow A$ such that $f \circ k = \mathrm{id}_B$. But then $f \circ k \circ f = f = f \circ \mathrm{id}_A$. But f being mono can be cancelled from the left to conclude that $k \circ f = \mathrm{id}_A$. □

8.1.9 Proposition *Every monomorphism in* **Set** *is regular.*

Proof. A monomorphism in **Set** is an injective function (see Theorem 3.3.3), so let $f : A \longrightarrow B$ be an injective function. Let C be the set of all pairs

$$\{(b, i) \mid b \in B,\ i = 0, 1\}$$

and impose an equivalence relation on these pairs forcing $(b, 0) = (b, 1)$ if and only if there is an $a \in A$ with $f(a) = b$ (and not forcing $(b, i) = (c, j)$ if b and c are distinct). Since f is injective, if such an a exists, there is only one. Let $g : B \longrightarrow C$ by $g(b) = (b, 0)$ and $h : B \longrightarrow C$ by $h(b) = (b, 1)$. Then clearly $g(b) = h(b)$ if and only if there is an $a \in A$ with $f(a) = b$. Now let $k : D \longrightarrow B$ with $g \circ k = h \circ k$. It must be that for all $x \in D$, there is an $a \in A$, and only one, such that $k(x) = f(a)$. If we let $l(x) = a$, then $l : D \longrightarrow A$ is the unique arrow with $f \circ l = k$. □

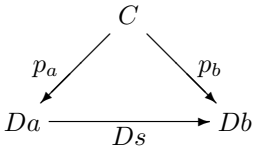
In contrast, not every monomorphism in **Cat** is regular. An example is given in [Barr and Wells, 1999], Section 9.1.

8.2 The general concept of limit

Products and equalizers are both examples of the general concept of limit.

8.2.1 Definition Let $\mathscr{G}$ be a graph and $\mathscr{C}$ be a category. Let $D : \mathscr{G} \to \mathscr{C}$ be a diagram in $\mathscr{C}$ with shape $\mathscr{G}$. A **cone** with base D is an object C of $\mathscr{C}$ together with a family $\{p_a\}$ of arrows of $\mathscr{C}$ indexed by the nodes of $\mathscr{G}$, such that $p_a : C \to Da$ for each node a of $\mathscr{G}$. The arrow p_a is the **component** of the cone at a.

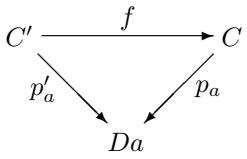
The cone is **commutative** if for any arrow $s : a \to b$ of $\mathscr{G}$, the diagram



commutes. Note: The diagram D is *not* assumed to commute.

Such a diagram will also be termed a commutative cone **over** D (or **to** D or with **base** D) with vertex C. We will write it as $\{p_a\} : C \to D$ or simply $p : C \to D$. It is clear that if $p : C \to D$ is a cone over D and $f : C' \to C$ is an arrow in $\mathscr{C}$, then there is a cone $p \circ f : C' \to D$ whose component at a is $p_a \circ f$. Moreover $p \circ f$ is commutative if p is. Note that a cone over a discrete diagram (see 6.1.3) is vacuously commutative.

8.2.2 Definition If $p' : C' \to D$ and $p : C \to D$ are cones, a **morphism** from p' to p is an arrow $f : C' \to C$ such that for each node a of $\mathscr{G}$, the diagram



commutes.

8.2.3 Definition A commutative cone over the diagram D is called **universal** if every other commutative cone over the same diagram has a unique arrow to it. A universal cone, if such exists, is called a **limit** of the diagram D.

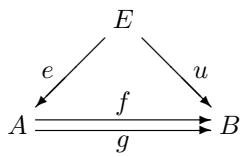
8.2.4 It is worth spelling out in some detail the meaning of a limit. To say that $p : C \to D$ is a limit means that there is given a family $p_a : C \to Da$, indexed by the nodes of $\mathscr{G}$, for which

L–1 Whenever $s : a \to b$ is an arrow of $\mathscr{G}$, then $Ds \circ p_a = p_b$.

L–2 If $p' : C' \to D$ is any other such family with the property that $Ds \circ p'_a = p'_b$ for every $s : a \to b$ in $\mathscr{G}$, then there is one and only one arrow $f : C' \to C$ such that for each node a of $\mathscr{G}$, $p_a \circ f = p'_a$.

8.2.5 Examples A limit cone over a finite discrete diagram is a product cone: here, L–1 and the commutativity condition $Ds \circ p'_a = p'_b$ in L–2 are vacuous.

Equalizers are also limits. Let $f, g : A \to B$ be parallel arrows in a category. An equalizer $e : E \to A$ is part of a cone



(8.1)

where $u = f \circ e = g \circ e$. This cone is commutative, and it is a limit cone if and only if e is an equalizer of f and g. Since u is determined uniquely by f and e (or by g and e), it was not necessary to mention it in Definition 8.1.2.

8.2.6 Equivalent definitions of limit There are two more equivalent ways to define limits. Let $D : \mathscr{G} \to \mathscr{C}$ be a diagram. We define the category cone(D) as follows. An object of this category is a commutative cone $\{p_i : C \to Di\}$ and an arrow is a morphism of cones (Definition 8.2.2). It is evident that the identity is a morphism and that the composite of two morphisms is another one. A terminal object in cone(D), if one exists, is a commutative cone over D to which every other commutative cone over D has a unique morphism. Thus we have shown that the existence of a limit is equivalent to the existence of a terminal object of cone(D).

The second equivalent construction of limits associates to D a functor we call cone$(-, D) : \mathscr{C}^{\mathrm{op}} \to \mathbf{Set}$. For an object C of $\mathscr{C}$, cone(C, D) is the set of commutative cones $p : C \to D$. If $f : C' \to C$, cone$(f, D) :$ cone$(C, D) \to$ cone(C', D) is defined by cone$(f, D)(p) = p \circ f$. It is straightforward to verify that this is a functor. A universal element of this functor is an object C and an element $p \in$ cone(C, D) such that for any C' and any element $p' \in$ cone(C', D) there is a unique arrow $f : C' \to C$ such that $p \circ f = p'$. But this is just the definition of a limit cone. (Compare the proof of Proposition 8.1.4.)

We have now sketched the proof of the following.

8.2.7 Theorem *The following three are equivalent for a diagram D in a category $\mathscr{C}$:*

(i) *a limit of D;*

(ii) *a terminal object of* cone(D)*;*

(iii) *a universal element of the functor* cone$(-, D)$*.*

In particular, one can see products and equalizers as terminal objects or as universal elements. One immediate consequence of this, in light of 5.7.17, is that a limit of a diagram is characterized uniquely up to a unique isomorphism in the same way that a product is.

8.2.8 Theorem *Let D be a diagram and $p : C \to D$ and $p' : C' \to D$ limits. Then there is a unique arrow $i : C \to C'$ such that for every node a of the shape graph of D, $p'_a \circ i = p_a$ and i is an isomorphism.*

8.2.9 Definition A category is said to **have all limits** or to be **complete** if every diagram has a limit. It is said to **have all finite limits** or to be **finitely complete** if every diagram whose domain is a finite graph has a limit.

The following theorem gives a useful criterion for the existence of all limits or all finite limits. Another way of getting all finite limits is given by Proposition 8.3.7.

8.2.10 Theorem *Suppose every set (respectively, every finite set) of objects of $\mathscr{C}$ has a product and every parallel pair of arrows has an equalizer. Then every diagram (respectively, every finite diagram) in $\mathscr{C}$ has a limit.*

The proof is omitted.

8.2.11 Corollary *A category $\mathscr{C}$ with the following properties*

(i) *$\mathscr{C}$ has a terminal object;*

(ii) *every pair of objects has a product; and*

(iii) *every parallel pair of arrows has an equalizer*

has all finite limits.

8.2.12 Definition A functor $F : \mathscr{C} \to \mathscr{D}$ **preserves limits** if it takes every limit cone in $\mathscr{C}$ to a limit cone in $\mathscr{D}$. It **preserves finite limits** if it takes every limit cone over a graph with a finite diagram in $\mathscr{C}$ to a limit cone in $\mathscr{D}$.

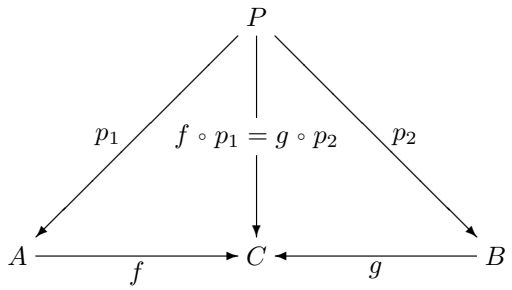
It follows from Corollary 8.2.11 that a functor that preserves terminal objects, products of pairs of objects, and equalizers of parallel pairs of arrows preserves all finite limits.

8.3 Pullbacks

Here is another example of a finite limit that will be important to us. Consider an object P and arrows $p_1 : P \longrightarrow A$ and $p_2 : P \longrightarrow B$ such that the diagram

$$\begin{array}{ccc} P & \xrightarrow{p_1} & A \\ {\scriptstyle p_2}\downarrow & & \downarrow{\scriptstyle f} \\ B & \xrightarrow[g]{} & C \end{array} \tag{8.2}$$

commutes. This does not appear directly to be a cone, because there is no arrow from the vertex P to C. It is understood to be a commutative cone with the arrow from P to C being the composite $f \circ p_1$, which by definition is the same as the composite $g \circ p_2$. It is common to omit such forced arrows in a cone. (Compare the discussion after Diagram (8.1).) It is more cone-like if we redraw the diagram as



However, the square shape of (8.2) is standard.

If this commutative cone is universal, then we say that P together with the arrows p_1 and p_2 is a **pullback** or **fiber product** of the pair. We also say that p_2 is the **pullback of f along g**, and that (8.2) is a **pullback diagram**. We often write $P = A \times_C B$, although this notation omits the arrows which are as important as the object C.

8.3.1 Example In **Set**, if $f : S \longrightarrow T$ and $g : U \longrightarrow T$ are functions, then the pullback

$$\begin{array}{ccc} P & \xrightarrow{p_1} & S \\ {\scriptstyle p_2}\downarrow & & \downarrow{\scriptstyle f} \\ U & \xrightarrow[g]{} & T \end{array} \tag{8.3}$$

is constructed by setting

$$P = \{(s, u) \mid f(s) = g(u)\}$$

with $p_1(s, u) = s$ and $p_2(s, u) = u$.

In one way this example is characteristic of pullbacks in any category with products: in any such category, P is a subobject of $S \times U$.

8.3.2 Example The inverse image of a function is a special case of a pullback. Suppose $g : S \longrightarrow T$ is a set function and $A \subseteq T$. Let $i : A \longrightarrow T$ be the inclusion. Let $g^{-1}(A) = \{x \in S \mid g(x) \in A\}$ be the inverse image of A, j be its inclusion in S, and h be the restriction of g to $g^{-1}(A)$. Then the following is a pullback diagram.

$$\begin{array}{ccc} g^{-1}(A) & \xrightarrow{h} & A \\ {\scriptstyle j}\downarrow & & \downarrow{\scriptstyle i} \\ S & \xrightarrow[g]{} & T \end{array} \tag{8.4}$$

Observe that Example 8.3.1 gives a general construction for pullbacks in sets, and the present example gives a construction for a special type of pullback, but this construction *is not a special case* of the construction in 8.3.1. By Theorem 8.2.8, there must be a unique function $u : \{(s, a) \mid g(s) = a\} \longrightarrow g^{-1}(A)$ for which $g(j(u(s, a))) = i(h(u(s, a)))$. By the definitions of the functions involved, $u(s, a)$ must be s.

The reader may wish to know the origin of the term 'fiber product'. Consider a function $f : S \longrightarrow T$. Of the many ways to think of a function, one is as determining a partition of S indexed by the elements of T. (See 2.6.12.) For $t \in T$, let

$$S_t = f^{-1}(t) = \{s \in S \mid f(s) = t\}$$

This is a family of disjoint subsets of S (some of which may be empty) whose union is all of S. This is sometimes described as a **fibration** of S and S_t is called the **fiber** over t. Now if $S \longrightarrow T$ and $U \longrightarrow T$ are two arrows and $S \times_T U$ is a pullback, then it is not hard to see that

$$S \times_T U = \bigcup \{S_t \times U_t \mid t \in T\}$$

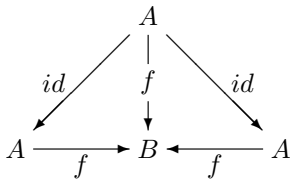
In other words, $S \times_T U$ is the fibered set whose fiber over any $t \in T$ is the product of the fibers. This is the origin of the term and of the notation. We will not use the term in these notes, but the notation has become standard and is too useful to abandon. Fibrations can be constructed for categories as well as sets; they are considered in Chapter 12 of [Barr and Wells, 1999].

Pullbacks can be used to characterize monomorphisms in a category.

8.3.3 Theorem *These three conditions are equivalent for any arrow $f : A \longrightarrow B$ in a category $\mathscr{C}$:*

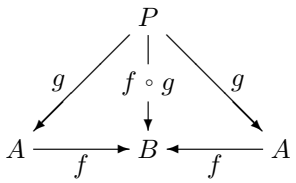
(a) *f is monic.*

(b) *The diagram*



is a limit cone.

(c) *There is an object P and an arrow $g : P \longrightarrow A$ for which*



is a limit cone.

(The operative condition in (c) is that the same arrow g appears on both slant lines.)
Another connection between pullbacks and monomorphisms is given by the following.

8.3.4 Proposition *In Diagram (8.2), if the arrow f is monic, then so is p_2.*

This proposition is summed up by saying, 'A pullback of a monic is a monic.'

8.3.5 Weakest precondition as pullback A widely used approach to program verification is to attach preconditions and postconditions to program fragments. An example is

$$\begin{array}{ll} 1. & \{\mathtt{X} < 3\} \\ 2. & \mathtt{X} := \mathtt{X} + 1; \\ 3. & \{\mathtt{X} < 24\} \end{array} \tag{8.5}$$

where `X` is an integer variable.

Statement 1 is called a **precondition** and statement 3 is a **postcondition**. The whole expression (8.5) is an **assertion** about the program: if the precondition is true before the program fragment is executed, the postcondition must be true afterward.

Clearly (8.5) is correct but a stronger assertion can be made. For the given postcondition, the weakest possible precondition is $\{\texttt{X} < \texttt{23}\}$. In a very general setting, there is a weakest precondition for every postcondition.

This can be placed in a categorical setting. Let D be the set of possible inputs and E be the set of possible outputs. Then the program fragment, provided it is deterministic and terminating, can be viewed as an arrow $f : D \to E$. Any condition C on a set X can be identified with the subset $X_0 = \{x \mid x \text{ satisfies } C\}$. In particular, the postcondition is a subset $S \subseteq E$ and the weakest precondition for that postcondition is the inverse image $f^{-1}(S)$, which is the unique subobject of D for which

$$\begin{array}{ccc} f^{-1}(S) & \rightarrowtail & D \\ \downarrow & & \downarrow \\ S & \rightarrowtail & E \end{array}$$

is a pullback. In the example, $f(x) = x + 1$, $S = \{x \mid x < 24\}$ and $f^{-1}(S) = \{x \mid x + 1 < 24\} = \{x \mid x < 23\}$.

This is easily checked in the case of **Set** and is a plausible point of view for other categories representing data and program fragments.

In the general case, one would expect that an assertion would be some special kind of subobject. For example, Manes [1986] requires them to be 'complemented' and Wagner [1987] requires them to be regular (see 8.1.7). Naturally, if one requires that assertions be subobjects with a certain property, one would want to work in a category in which pullbacks of such subobjects also had the property. In a topos (toposes will be discussed in Chapter 11), for example, pullbacks of complemented subobjects are complemented, and all monomorphisms are regular.

8.3.6 Constructing all finite limits revisited Corollary 8.2.11 described a class of limits whose existence guarantees the existence of all finite limits. A similar construction is possible for pullbacks.

8.3.7 Proposition *A category that has a terminal object and all pullbacks has all finite limits.*

8.4 Coequalizers

In 8.1.2 we introduced the notion of equalizer. The dual notion is called coequalizer. Explicitly, consider $f, g : A \rightrightarrows B$. An arrow $h : B \to C$ is called a **coequalizer** of f and g provided $h \circ f = h \circ g$ and for any arrow $k : B \to D$ for which $k \circ f = k \circ g$, there is a unique arrow $l : C \to D$ such that $l \circ h = k$.

The way to think about the coequalizer of f and g is as the quotient object made by forcing f and g to be equal. In the category of sets, this observation becomes the construction that follows.

8.4.1 Coequalizers in Set Let $f, g : S \rightrightarrows T$ be a pair of arrows in the category of sets. The conditions a coequalizer $h : T \to U$ must satisfy are that $h \circ f = h \circ g$ and that given any arrow $k : T \to V$, the equation $k \circ f = k \circ g$ implies the existence of a unique arrow $l : U \to V$ such that $l \circ h = k$. (If such an arrow l exists, then necessarily $k \circ f = k \circ g$.)

The pair f and g determine a relation $R \subseteq T \times T$ which consists of

$$\{(f(s), g(s)) \mid s \in S\}$$

The equation $k \circ f = k \circ g$ is equivalent to the statement that $k(t_1) = k(t_2)$ for all $(t_1, t_2) \in R$.

In general R is not an equivalence relation, but it can be completed to one by forming the reflexive, symmetric and transitive closures of R in that order. The reflexive, symmetric closure is the union $R_1 = R \cup \Delta \cup R^{\text{op}}$ where $\Delta = \{(t, t) \mid t \in T\}$ and $R^{\text{op}} = \{(t_2, t_1) \mid (t_1, t_2) \in R\}$. If we inductively define R_{i+1} to be the set

$$\{(t_1, t_2) \mid \exists t \in T((t_1, t) \in R_1 \text{ and } (t, t_2) \in R_i)\}$$

for each $i \in \mathbf{N}$ and define $\widetilde{R} = \bigcup R_n$, then $\widetilde{R}$ is the transitive closure of R_1 and is the least equivalence relation containing R.

When S and T are finite, the least equivalence relation of the previous paragraph can be computed efficiently by Warshall's Algorithm ([Sedgewick, 1983], p. 425).

Let U be the set of equivalence classes modulo the equivalence relation $\widetilde{R}$ and let $h : T \longrightarrow U$ be the function that sends an element of T to the class that contains that element.

8.4.2 Proposition *The arrow h is a coequalizer of f and g.*

Proof. Since $R \subseteq \widetilde{R}$, the fact that $h(t_1) = h(t_2)$ for all $(t_1, t_2) \in \widetilde{R}$ implies, in particular, that the same equation is satisfied for all $(t_1, t_2) \in R$; this means that for all $s \in S$, $h(f(s)) = h(g(s))$, so that $h \circ f = h \circ g$.

Now let $k : T \longrightarrow V$ satisfy $k \circ f = k \circ g$. We claim that for $(t_1, t_2) \in \widetilde{R}$, $k(t_1) = k(t_2)$. In fact, this is true for $(t_1, t_2) \in R$. It is certainly true for $t_1 = t_2$, i.e. $(t_1, t_2) \in \Delta$ implies that $k(t_1) = k(t_2)$. Since $k(t_1) = k(t_2)$ implies that $k(t_2) = k(t_1)$, we now know that the assertion is true for $(t_1, t_2) \in R_1$. Since $k(t_1) = k(t_2)$ and $k(t_2) = k(t_3)$ imply $k(t_1) = k(t_3)$, one may show by induction that the assertion is true for all the R_n and hence for $\widetilde{R}$. This shows that k is constant on equivalence classes and hence induces a function $l : U \longrightarrow V$ such that $l \circ h = k$. Since h is surjective, l is unique. □

Sums and coequalizers are used in [Rydeheard and Burstall, 1985] to provide a theoretical basis for the theory of unification used in logic programming. See also [Barr and Wells, 1999], Chapter 9, [Rydeheard and Burstall, 1986] and [Goguen, 1988].

8.4.3 Regular epimorphisms In Proposition 8.1.5, it is asserted that all equalizers are monomorphisms. The dual is of course also true; all coequalizers are epimorphisms. The converse is not true; not every epimorphism is a coequalizer of some pair of arrows. For example, in the category of monoids, the inclusion of the nonnegative integers into the set of all integers is an epimorphism but is not the coequalizer of any pair of arrows. Those epimorphisms that are the coequalizer of some pair of arrows into their domain are special and this is marked by giving them a special name: they are called **regular epimorphisms**.

In the category of sets, every epimorphism is regular It follows that in the category of sets, a pullback of a regular epi is regular. In contrast to this, let $\mathbf{N}$ denote the monoid of natural numbers on addition and $\mathbf{Z}$ the monoid of integers on addition. Then the inclusion function is an epimorphism (in the category of monoids) which is not regular. (But every surjective epimorphism in the category of monoids is regular.)

A category is called a **regular category** if it has finite limits, if every parallel pair of arrows has a coequalizer, and if the arrow opposite a regular epimorphism in a pullback diagram is a regular epimorphism. A functor between regular categories is called a **regular functor** if it preserves finite limits and regular epis.

It is an old theorem (stated in different language) that all categories of models of FP sketches, in other words, all varieties of multisorted algebraic structures, are regular ([Barr and Wells, 1985], Theorem 1 of Section 8.4). Because of such examples, regular categories have seen considerable theoretical study, and regular epimorphisms have played a larger role in category theory than regular monos have.

8.5 Cocones

Just as there is a general notion of limit of which products and equalizers are special cases, so there is a general notion of colimit.

8.5.1 Definition A **cocone** is a cone in the dual graph. Spelling the definition out, it is a diagram $D : \mathscr{G}_0 \longrightarrow \mathscr{G}$, a node g of $\mathscr{G}$ together with a family $u_a : Da \longrightarrow g$ of arrows indexed by the objects of $\mathscr{G}_0$. The diagram is called the **base** of the cocone. If the diagram is discrete, it is called a **discrete cocone**.

A cocone in a category is called **commutative** if it satisfies the dual condition to that of commutative cone. Explicitly, if the diagram is $D : \mathscr{G} \longrightarrow \mathscr{C}$ and the cone is $\{u_a : Da \longrightarrow C \mid a \in \mathrm{Ob}(\mathscr{G})\}$, then what

is required is that for each arrow $s : a \longrightarrow b$ of $\mathscr{G}$, $u_b \circ Ds = u_a$. The commutative cocone is called a **colimit** cocone if it has a unique arrow to every other commutative cocone with the same base.

8.5.2 Cocompleteness A category **has all colimits** or is **cocomplete** if every diagram in the category has a colimit. It **has finite colimits** or is **finitely cocomplete** if every diagram with a finite graph has a colimit.

Categories of algebraic structures are complete and cocomplete. If we disallowed the empty semigroup (see 3.2.1) we would have to say that the category of semigroups is 'complete and cocomplete except that it does not have an initial object'. We would also have to say that the intersection of any set of subsemigroups of a semigroup is either a subsemigroup or empty.

8.5.3 Pushouts For example, dual to the notion of pullback is that of pushout. In detail, a commutative square

$$\begin{array}{ccc} C & \xrightarrow{f} & A \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle q_1} \\ B & \xrightarrow[q_2]{} & Q \end{array} \tag{8.6}$$

is called a **pushout** if for any object R and any pair of arrows $r_1 : A \longrightarrow R$ and $r_2 : B \longrightarrow R$ for which $r_1 \circ f = r_2 \circ g$ there is a unique arrow $r : Q \longrightarrow R$ such that $r \circ q_i = r_i$, $i = 1, 2$.

8.5.4 Pushouts as amalgamations If the effect of coequalizers is to force identifications, that of pushouts is to form what are called amalgamated sums. We illustrate this with examples.

First, consider the case of a set S and three subsets S_0, S_1 and S_2 with $S_0 = S_1 \cap S_2$ and $S = S_1 \cup S_2$. Then the diagram

$$\begin{array}{ccc} S_0 & \longrightarrow & S_1 \\ \downarrow & & \downarrow \\ S_2 & \longrightarrow & S \end{array}$$

with all arrows inclusion, is a pushout. (It is a pullback as well; such diagrams are often referred to as **Doolittle diagrams**.) The definition of pushout translates to the obvious fact that if one is given functions $f_1 : S_1 \longrightarrow T$ and $f_2 : S_2 \longrightarrow T$ and $f_1|S_0 = f_2|S_0$, then there is a unique function $f : S \longrightarrow T$ with $f|S_1 = f_1$ and $f|S_2 = f_2$.

Now consider a slightly more general situation. We begin with sets S_0, S_1 and S_2 and functions $g_1 : S_0 \longrightarrow S_1$ and $g_2 : S_0 \longrightarrow S_2$. If g_1 and g_2 are injections, then, up to isomorphism, this may be viewed as the same as the previous example. Of course, one cannot then form the union of S_1 and S_2, but rather first the disjoint sum and then, for $s \in S_0$, identify the element $g_1(s) \in S_1$ with $g_2(s) \in S_2$. In this case, the pushout is called an **amalgamated sum** of S_1 and S_2.

More generally, the g_i might not be injective and then the amalgamation might identify two elements of S_1 or of S_2, but the basic idea is the same; *pushouts are the way you identify part of one object with a part of another.*

8.5.5 Equations, equalizers and coequalizers One way of thinking about an equalizer is as the largest subobject on which an equation or set of equations is true. A coequalizer, by contrast, is the least destructive identification necessary to force an equation to be true on the equivalence classes.

Here is an instructive example. There are two ways of defining the rational numbers. They both begin with the set $\mathbf{Z} \times \mathbf{N}^+$ of pairs of integers (a, b) for which $b > 0$. For the purpose of this illustration, we will write (a/b) instead of (a, b). In the first, more familiar, construction, we identify (a/b) with (c/d) when $a * d = b * c$. One way of describing this is as the coequalizer of two arrows

$$T \rightrightarrows \mathbf{Z} \times \mathbf{N}^+$$

where T is the set of all $(a, b, c, d) \in \mathbf{Z} \times \mathbf{N}^+ \times \mathbf{Z} \times \mathbf{N}^+$ such that $a * d = b * c$ and the first arrow sends (a, b, c, d) to (a/b) while the second sends it to (c/d). The effect of the coequalizer is to identify (a/b) with (c/d) when $a * d = b * c$.

The second way to define the rationals is the set of pairs that are relatively prime (that is, in lowest terms). This can be realized as an equalizer as follows. Let $\gcd : \mathbf{Z} \times \mathbf{N}^+ \to \mathbf{N}^+$ take a pair of numbers to their positive greatest common divisor. Let $1 \circ \langle\rangle : \mathbf{Z} \times \mathbf{N}^+ \to \mathbf{N}^+$ denote the function that is constantly 1. Then an equalizer of gcd and $1 \circ \langle\rangle$ is exactly the set of relatively prime pairs.

9. Adjoints

Adjoints are about the most important idea in category theory, except perhaps for the closely related notion of representable functors. Many constructions made earlier in these notes are examples of adjoints, as we will describe. Moreover, toposes are most conveniently described in terms of adjoints. Section 9.1 revisits the concept of free monoids, describing them in a way which suggests the general definition of adjoint. Adjoints are described and some basic properties developed in Sections 9.2 and 9.3.

More detail concerning adjoints can be found in [Barr and Wells, 1985] and [Mac Lane, 1971]. Diers [1980a], [1980b] describes a generalization which includes the initial families of finite-discrete sketches. Hagino [1987a], [1987b] has a general approach to type constructors for functional programming languages based on adjoints (see also [Chen and Cockett, 1989]).

9.1 Free monoids

In 4.1.15, we gave the universal property of the free monoid. We now state it more carefully than we did there, paying closer attention to the categories involved. Note that when we speak of a sub*set* of a monoid, we are mixing two things. A monoid is not just a set and most of its subsets are not submonoids. To describe a monoid, you must give three data: the set of elements, the operation and the identity element. From this point of view the phrase 'subset of a monoid' does not make sense. It is actually a subset of the underlying set of the monoid. Insistence on this is not pointless pedantry; it is the key to understanding the situation.

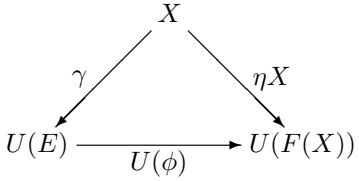
Let **Mon** denote the category of monoids and let $U : \mathbf{Mon} \longrightarrow \mathbf{Set}$ denote the underlying set functor. We define a monoid M to be a triple $(UM, \cdot, 1)$, where UM is a set called the **underlying set** of M, $\cdot$ is a binary operation on UM and $1 \in UM$ is the identity element for that operation. A homomorphism from a monoid M to a monoid N is a function $f : UM \longrightarrow UN$ which preserves the operation and the identity element. The underlying functor is defined on homomorphisms by $Uf = f$.

9.1.1 Characterization of the free monoid Given a set X, the free monoid $F(X) = (X^*, \cdot, \langle\rangle)$ is characterized by the property given in the following proposition; the property is called the **universal mapping property** of the free monoid. In the proposition, $\eta X : X \longrightarrow X^* = UF(X)$ takes $x \in X$ to the string $\langle x \rangle$ of length 1. We systematically distinguish the free monoid $F(X)$ from its underlying set, the Kleene closure $X^* = U(F(X))$.

9.1.2 Proposition *Let X be a set. For any monoid M and any function $u : X \longrightarrow U(M)$, there is a unique monoid homomorphism $g : F(X) \longrightarrow M$ such that $u = Ug \circ \eta X$.*

Proof. Define $g(\langle\rangle)$ to be the identity element of M, $g(\langle x \rangle) = u(x)$ for $x \in X$, and $g(\langle x_1 \cdots x_n \rangle)$ to be the product $u(x_1) \cdot \cdots \cdot u(x_n)$ in M. That makes g a monoid homomorphism and $u = Ug \circ \eta X$.

This proposition says ηX is a universal element for the functor $\mathrm{Hom}(X, U-)$ that takes a monoid M to $\mathrm{Hom}(X, U(M))$ and a monoid homomorphism $f : M \longrightarrow N$ to $\mathrm{Hom}(X, U(f)) : \mathrm{Hom}(X, U(M)) \longrightarrow \mathrm{Hom}(X, U(N))$. It follows that the universal mapping property characterizes the free monoid up to a unique isomorphism. Precisely, let X be a set and $\gamma : X \longrightarrow U(E)$ a function to the underlying set of some monoid E with the property that if $u : X \longrightarrow U(M)$, where M is any monoid, there is a unique monoid homomorphism $f : E \longrightarrow M$ such that $u = U(f) \circ \gamma$. We now show that there is a unique isomorphism $\phi : E \longrightarrow F(X)$ for which



commutes. As we have noted above, ηX is a universal element of the functor $\mathrm{Hom}(X, U(-))$. The assumption on γ says that it is also a universal element for the same functor. Now Corollary 5.7.17

says there is a unique isomorphism $\phi : E \longrightarrow F(X)$ for which $\mathrm{Hom}(X, U(\phi))(\gamma) = \eta X$. The result follows because

$$\mathrm{Hom}(X, U(\phi))(\gamma) = U(\phi) \circ \gamma$$

by definition.

9.1.3 The free monoid functor Each set X generates a free monoid $F(X)$. In 4.1.13, we extended this to a functor $F : \mathbf{Set} \longrightarrow \mathbf{Mon}$. In this section, we will prove this (as part of the proof of Proposition 9.1.4 below) using only the universal mapping property of free monoids; thus the argument will work in complete generality.

Let ηY denote the arrow from Y into $Y^* = UFY$ described in 9.1.1 and let $f : X \longrightarrow Y$ be a function. Then $\eta Y \circ f : X \longrightarrow Y^*$ is a function from X into the set underlying a monoid. The universal property of $F(X)$ gives a unique monoid homomorphism $F(f) : F(X) \longrightarrow F(Y)$ such that if $f^* = UF(f)$, then

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle \eta X}\downarrow & & \downarrow{\scriptstyle \eta Y} \\ X^* & \xrightarrow[f^*]{} & Y^* \end{array} \tag{9.1}$$

commutes.

9.1.4 Proposition *$F : \mathbf{Set} \longrightarrow \mathbf{Mon}$ is a functor and, for any set X, ηX is the component at X of a natural transformation $\eta : \mathrm{id}_{\mathbf{Set}} \longrightarrow U \circ F$.*

Proof. Once F is shown to be a functor, that η is a natural transformation will follow from Diagram (9.1).

First note that if $\mathrm{id} : X \longrightarrow X$ is the identity, then $F(\mathrm{id})$ is the unique arrow $h : F(X) \longrightarrow F(X)$ such that

$$\begin{array}{ccc} X & \xrightarrow{\mathrm{id}} & X \\ {\scriptstyle \eta X}\downarrow & & \downarrow{\scriptstyle \eta X} \\ X^* & \xrightarrow[Uh]{} & X^* \end{array}$$

commutes. But if we replace h in the diagram above by id_{FX}, the diagram still commutes. The uniqueness property of the free monoid implies that $h = \mathrm{id}_{FX}$.

Similarly, if $g : Y \longrightarrow Z$ is a function, the commutativity of both squares in

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle \eta X}\downarrow & & \downarrow{\scriptstyle \eta Y} \\ X^* & \xrightarrow[f^*]{} & Y^* \end{array} \qquad \begin{array}{ccc} Y & \xrightarrow{g} & Z \\ {\scriptstyle \eta Y}\downarrow & & \downarrow{\scriptstyle \eta Z} \\ Y^* & \xrightarrow[g^*]{} & Z^* \end{array}$$

implies that

$$\begin{array}{ccc} X & \xrightarrow{g \circ f} & Z \\ {\scriptstyle \eta X}\downarrow & & \downarrow{\scriptstyle \eta Z} \\ X^* & \xrightarrow[g^* \circ f^*]{} & Z^* \end{array}$$

commutes. But $g^* \circ f^* = UF(g) \circ UF(f) = U(F(g) \circ F(f))$ since U is a functor. This means that the arrow $F(g) \circ F(f)$ satisfies the same equation that characterizes $F(g \circ f)$ uniquely and hence that they are equal. This shows that F is a functor and that η is a natural transformation. □

Another example of functors F and U satisfying this is the free category given by a graph as described in 2.6.16. This very same proof shows that the free category construction is really the object part of a functor.

9.2 Adjoints

The relationship between the free monoid functor and the underlying set functor is an example of a very general situation, an adjunction, defined below.

In this section, we have several notational shortcuts to prevent clutter of parentheses and composition circles. This notation is quite standard in the literature on adjoints. For example, an expression $UFUA$ is shorthand for $(U \circ F \circ U)(A)$, which is the same as $U(F(U(A)))$.

9.2.1 Definition Let $\mathscr{A}$ and $\mathscr{B}$ be categories. If $F : \mathscr{A} \to \mathscr{B}$ and $U : \mathscr{B} \to \mathscr{A}$ are functors, we say that F is **left adjoint** to U and U is **right adjoint** to F provided there is a natural transformation $\eta : \mathrm{id} \to UF$ such that for any objects A of $\mathscr{A}$ and B of $\mathscr{B}$ and any arrow $f : A \to UB$, there is a unique arrow $g : FA \to B$ such that

$$\begin{array}{ccc} A & \xrightarrow{\eta A} & UFA \\ & \searrow^{f} & \downarrow{\scriptstyle Ug} \\ & & UB \end{array} \tag{9.2}$$

commutes.

This definition asserts that there is a functional way to convert any arrow $f : A \to UB$ to an arrow $g : FA \to B$ in such a way that g solves the equation $f = U(?) \circ \eta A$, and that the solution is unique.

The property of η given in the last sentence of Definition 9.2.1 is called its **universal mapping property**. The existence of the unique arrow $g : FA \to B$ such that $f = Ug \circ \eta A$ is the map-lifting property of Section 4.1.15. Just as in the discussion after Proposition 9.1.2, for each object A, the arrow ηA is a universal element for the functor $\mathrm{Hom}(A, U(-))$.

It is customary to write $F \dashv U$ to denote the situation described in the definition. The data (F, U, η) constitute an **adjunction**. The transformation η is called the **unit** of the adjunction.

In some texts the adjunction is written as a rule of inference, like this:

$$\frac{A \to UB}{FA \to B} \tag{9.3}$$

The definition appears to be asymmetric in F and U. This is remedied in the next proposition, whose proof is deferred to the next section, after the proof of Theorem 9.3.2.

9.2.2 Proposition *Suppose $F : \mathscr{A} \to \mathscr{B}$ and $U : \mathscr{B} \to \mathscr{A}$ are functors such that $F \dashv U$. Then there is a natural transformation $\epsilon : FU \to \mathrm{id}_{\mathscr{B}}$ such that for any $g : FA \to B$, there is a unique arrow $f : A \to UB$ such that*

$$\begin{array}{ccc} & & FUB \\ & \nearrow^{Ff} & \downarrow{\scriptstyle \epsilon B} \\ FA & \xrightarrow[g]{} & B \end{array} \tag{9.4}$$

The transformation ϵ is called the **counit** of the adjunction.

It is an immediate consequence of categorical duality that this proposition is reversible and the adjunction is equivalent to the existence of either natural transformation η or ϵ with its appropriate universal mapping property.

9.2.3 Examples of adjoints We have of course the example of free monoids that introduced this chapter. In general, let $\mathscr{C}$ be a category of sets with structure and functions which preserve the structure, and let $U : \mathscr{C} \longrightarrow \mathbf{Set}$ be the underlying set functor. If U has a left adjoint F and S is a set, then $F(S)$ is the **free structure** on S. This description fits structures you may know about, such as free groups, free Abelian groups and free rings.

We now give three related examples that illustrate how widespread adjoints are.

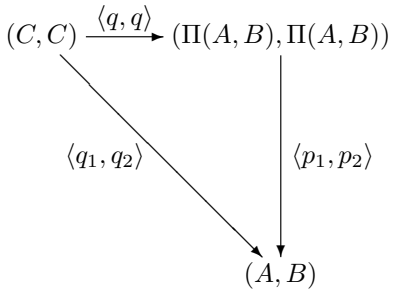
9.2.4 Example Let $\mathscr{C}$ be a category. Consider the category $\mathscr{C} \times \mathscr{C}$ which has as objects pairs of objects (A, B) of $\mathscr{C}$ and in which an arrow $(A, B) \longrightarrow (A', B')$ is a pair (f, g) of arrows $f : A \longrightarrow A'$ and $g : B \longrightarrow B'$. There is a functor $\Delta : \mathscr{C} \longrightarrow \mathscr{C} \times \mathscr{C}$ given by $\Delta(A) = (A, A)$ and $\Delta(f) = (f, f)$. Let us see what a right adjoint to this functor is.

Assuming there is a right adjoint Π to Δ, there should be an arrow we call

$$\langle p_1, p_2 \rangle : \Delta\Pi(A, B) = (\Pi(A, B), \Pi(A, B)) \longrightarrow (A, B)$$

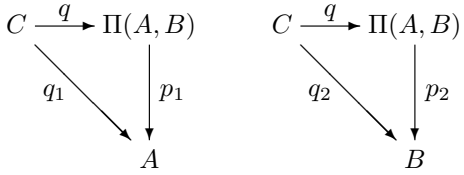
(this is the counit of the adjunction) with the following universal property:

PP For any object C of $\mathscr{C}$ and any arrow $\langle q_1, q_2 \rangle : \Delta C = (C, C) \longrightarrow (A, B)$ there is a unique arrow $q : C \longrightarrow \Pi(A, B)$ such that



commutes.

A diagram in a product category commutes if and only if each component does. Breaking the triangle into components the adjunction asserts the existence of a unique map $q : C \longrightarrow \Pi(A, B)$ such that each of the triangles



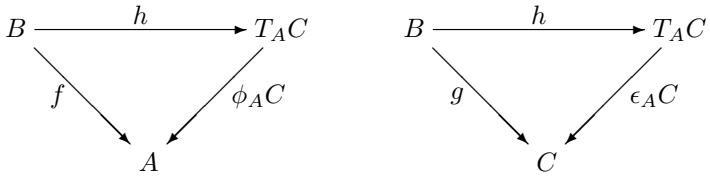
commutes. If you write $A \times B$ instead of $\Pi(A, B)$ you will recover the categorical definition of the product. Thus a right adjoint to Δ is just a functor $\Pi : \mathscr{C} \times \mathscr{C} \longrightarrow \mathscr{C}$ that chooses a product for each pair of objects of $\mathscr{C}$. Such a functor Π is called a **binary product functor**.

9.2.5 Example Now let us suppose we are given a binary product functor Π. We can fix an object A of $\mathscr{C}$ and consider the functor denoted $- \times A : \mathscr{C} \longrightarrow \mathscr{C}$ whose value at an object B is the object $B \times A$ and at an arrow $f : B \longrightarrow C$ is the arrow $f \times \mathrm{id}_A : B \times A \longrightarrow C \times A$ (see 6.2.17).

A right adjoint to the functor $- \times A$, if one exists, can be described as follows. Denote the value at C of this adjoint by $R_A(C)$. Then there is an arrow $e : R_A(C) \times A \longrightarrow C$ with the universal property that for any arrow $f : B \times A \longrightarrow C$, there is a unique arrow we may call $\lambda f : B \longrightarrow R_A(C)$ such that $e \circ (\lambda f \times A) = f$. But this is precisely the universal property that describes the exponential $[A \longrightarrow C]$ (see 7.1.3). The counit e is the arrow eval defined there. The essential uniqueness of adjoints (see 9.3.4 below) implies that this adjunction property determines the exponential, if it exists, up to isomorphism.

Thus the two main defining properties defining cartesian closed categories, namely the existence of binary products and the existence of exponentials, can be (and commonly are) described in terms of adjoints.

9.2.6 Example We can describe the functor $- \times A$ of 9.2.5 in a slightly different way. For a category $\mathscr{C}$ and object A of $\mathscr{C}$ we described the slice category $\mathscr{C}/A$ in 2.6.10 and functor $U_A : \mathscr{C}/A \to \mathscr{C}$ in 4.1.12. If U_A has a right adjoint $P_A : \mathscr{C} \to \mathscr{C}/A$, then P_A must associate to each object C of $\mathscr{C}$ an object $\phi_A C = P_A(C) : T_A C \to A$ of $\mathscr{C}/A$ and an arrow (the counit) $\epsilon_A C : T_A C \to C$. This object and arrow must have the universal mapping property that for any other object $f : B \to A$ of $\mathscr{C}/A$, and any arrow $g : B \to C$ there is a unique arrow $h : B \to T_A C$ such that



commute. The left triangle must commute in order to have an arrow in $\mathscr{C}/A$ and the right hand triangle must commute for its universal mapping property. It is evident from this description that $T_A C$ is $C \times A$ and $\epsilon_A C$ and $\phi_A C$ are the first and second projections. Thus $P_A(C)$ is the object $p_2 : C \times A \to A$ of $\mathscr{C}/A$.

9.2.7 Example For any set A with other than one element, the functor $- \times A : \mathbf{Set} \to \mathbf{Set}$ defined in 9.2.5 does not have a left adjoint in **Set**. For if it has a left adjoint F, then by definition of left adjoint there is for any set X an arrow $\eta X : X \to FX \times A$ with the property that for any function $f : X \to Y \times A$ there is a unique function $g : FX \to Y$ for which $(g \times A) \circ \eta X = f$. Now take $Y = 1$, the terminal object (any one element set). There is only one function $g : FX \to 1$, so there can be only one function $f : X \to 1 \times A \cong A$. If A has more than one element and X is non-empty, this is a contradiction.

9.2.8 Adjoints to the inverse image functor Here is another example of an adjoint. If S is a set, the set of subsets of S is a poset ordered by inclusion. This becomes a category in the usual way. That is, if S_0 and S_1 are subsets of S, then there is exactly one arrow $S_0 \to S_1$ if and only if $S_0 \subseteq S_1$. You should not view this arrow as representing a function, but just a formal arrow that represents the inclusion. We will call this category $\mathrm{Sub}(S)$.

If $f : S \to T$ is a function, then with any subset $T_0 \subseteq T$, we get a subset, denoted

$$f^{-1}(T_0) = \{s \in S \mid f(s) \in T_0\}$$

which is called the **inverse image** under f of T_0. If $T_0 \subseteq T_1$, then $f^{-1}(T_0) \subseteq f^{-1}(T_1)$. This means that f^{-1} is a functor from $\mathrm{Sub}(T) \to \mathrm{Sub}(S)$. This functor turns out to have both left and right adjoints.

The left adjoint is a familiar one (to mathematicians at least), the so-called **direct image.** For $S_0 \subseteq S$, let

$$f_*(S_0) = \{f(s) \mid s \in S_0\}$$

Then $f_*(S_0) \subseteq T_0$ if and only if $S_0 \subseteq f^{-1}(T_0)$, which is just the statement that the direct image is left adjoint to the inverse image.

The right adjoint of the inverse image is usually denoted $f_!$ and is defined by saying that $t \in f_!(S_0)$ if and only if $f^{-1}(\{t\}) \subseteq S_0$. Another way of saying this is that *every* element of the inverse image of t is in S_0, while $t \in f_*(S)$ if and only if *some* element of the inverse image of t is in S_0.

We discussed these constructions in 4.1.19 and 4.1.20 for **Set**.

9.2.9 Adjoints in posets Example 9.2.8 is a special case of an adjunction between ordered or even preordered sets. Let P and Q be preordered sets, that is categories in which there is at most one arrow between any pair of objects. We will use $\leq$ for the preorder relation, and the reader may find it more natural to think of this relation as a partial ordering (antisymmetric as well as reflexive and transitive). If $f : P \to Q$ and $u : Q \to P$ are functors, that is, order preserving functions, then $f \dashv u$ means, when translated into poset language (where uniqueness is not a consideration, since there is at most one arrow between any two objects), that

$$x \leq (u \circ f)(x) \text{ for all } x \in X \tag{9.5}$$

and

$$x \leq u(y) \text{ implies } f(x) \leq y \tag{9.6}$$

for all $x \in X$ and $y \in Y$. We claim that these two conditions together are equivalent to the condition

$$f(x) \leq y \text{ if and only if } x \leq u(y) \tag{9.7}$$

In fact, assuming (9.5) and (9.6), then if $f(x) \leq y$, we have $x \leq (u \circ f)(x) \leq u(y)$; condition (9.7) follows from this and (9.6). Conversely if (9.7) holds, then (9.6) is immediate and $f(x) \leq f(x)$ implies that $x \leq (u \circ f)(x)$. This is a special case of the Hom set adjunction of the next section (Theorem 9.3.2).

A very common situation involves a pair of contravariant, that is order reversing, functors. In that case, we can dualize either P or Q. Suppose we do the former. Then we have a pair of functors $f : P^{\mathrm{op}} \to Q$ and $u : Q \to P^{\mathrm{op}}$. If $f \dashv u$, then, translating this condition back to P, we see that $f(x) \leq y$ if and only if $u(y) \leq x$. Such a pair of contravariant functors is often called a **Galois connection** between P and Q, named after the first one produced by Galois.

If you know Galois theory of fields, then you know that E is a Galois extension of F, then there is a one-one order reversing correspondence between subfields of E that include F and subgroups of the Galois group of E over F. This particular Galois correspondence is an equivalence and some people prefer to restrict the use of the term 'Galois connection' to the case of equivalence, while others use it as we have and call the other a Galois equivalence.

9.2.10 Example Let $\mathbf{R}$ and $\mathbf{Z}$ denote the ordered sets of real numbers and integers, respectively, considered as categories. Then the left adjoint to the inclusion $\mathbf{Z} \subseteq \mathbf{R}$ is the ceiling function and the right adjoint is the floor function.

9.3 Further topics on adjoints

9.3.1 Hom set adjointness There is an alternative formulation of adjointness which is often found in the categorical literature.

9.3.2 Theorem *If $\mathscr{A}$ and $\mathscr{B}$ are categories and $U : \mathscr{B} \to \mathscr{A}$ and $F : \mathscr{A} \to \mathscr{B}$ are functors, then $F \dashv U$ if and only if* $\mathrm{Hom}(F-,-)$ *and* $\mathrm{Hom}(-,U-)$ *are naturally isomorphic as functors $\mathscr{A}^{\mathrm{op}} \times \mathscr{B} \to$* **Set**.

Proof. Let $F \dashv U$, and let A and B be objects of $\mathscr{A}$ and $\mathscr{B}$ respectively. Define $\beta_{A,B} : \mathrm{Hom}(FA, B) \to \mathrm{Hom}(A, UB)$ by $\beta_{A,B}(g) = Ug \circ \eta A$, and $\gamma_{A,B} : \mathrm{Hom}(A, UB) \to \mathrm{Hom}(FA, B)$ by requiring that $\gamma_{A,B}(f)$ be the unique arrow g such that $f = Ug \circ \eta A$ given by Definition 9.2.1. Then $\gamma_{A,B}(\beta_{A,B}(g)) = g$ by the uniqueness requirement of Definition 9.2.1, and $\beta_{A,B}(\gamma_{A,B}(f)) = f$ by definition of $\beta_{A,B}$ and $\gamma_{A,B}$. Thus $\beta_{A,B}$ is an isomorphism with inverse $\gamma_{A,B}$. The proof of naturality is omitted.

To go in the other direction, suppose we have the natural isomorphism. Then let A be an arbitrary object of $\mathscr{A}$ and $B = FA$. We then get $\mathrm{Hom}(FA, FA) \cong \mathrm{Hom}(A, UFA)$. Let $\eta A \in \mathrm{Hom}(A, UFA)$ be the arrow that corresponds under the isomorphism to the identity arrow of FA. Now for an arrow $f : A \to UB$, that is $f \in \mathrm{Hom}(A, UB)$, let $g \in \mathrm{Hom}(FA, B)$ be the arrow that corresponds under the isomorphism. Naturality of the isomorphism implies that we have a commutative diagram

$$\begin{array}{ccc} \mathrm{Hom}(FA, FA) & \xrightarrow{\cong} & \mathrm{Hom}(A, UFA) \\ \Big\downarrow{\scriptstyle \mathrm{Hom}(FA, g)} & & \Big\downarrow{\scriptstyle \mathrm{Hom}(A, Ug)} \\ \mathrm{Hom}(FA, B) & \xrightarrow[\cong]{} & \mathrm{Hom}(A, UB) \end{array}$$

If we follow the identity arrow of FA around the clockwise direction, we get first the arrow ηA by definition, and then $\mathrm{Hom}(A, Ug)(\eta A) = Ug \circ \eta A$. In the other direction, we get $\mathrm{Hom}(FA, g)(\mathrm{id}) = g \circ \mathrm{id} = g$ and that corresponds under the isomorphism to f. Thus we conclude that $f = Ug \circ \eta A$. As for the uniqueness, if $f = Uh \circ \eta A$, then both g and h correspond to f under the isomorphism, so $g = h$. □

9.3.3 Proof of Proposition 9.2.2 The hypotheses of the above theorem are symmetric in F and U and therefore the conclusion must be too. Thus categorical duality implies the existence of ϵ and the requisite universal mapping property. □

9.3.4 Uniqueness of adjoints If $U : \mathscr{B} \to \mathscr{A}$ is a functor, then a left adjoint to U, if one exists, is unique up to natural isomorphism. The reason is that if both F and F' are left adjoint to U, then for any object A of $\mathscr{A}$, the Hom functors $\mathrm{Hom}_{\mathscr{B}}(FA, -)$ and $\mathrm{Hom}_{\mathscr{B}}(F'A, -)$ are each naturally isomorphic to $\mathrm{Hom}_{\mathscr{A}}(A, U-)$ and hence to each other. It follows from the Yoneda embedding, Theorem 5.7.6, that $FA \cong F'A$. The naturality of the latter isomorphism follows from the next theorem.

9.3.5 Theorem *Let $\mathscr{A}$ and $\mathscr{B}$ be categories and $U : \mathscr{B} \to \mathscr{A}$ be a functor. Suppose for each object A of $\mathscr{A}$ there is an object FA of $\mathscr{B}$ such that $\mathrm{Hom}_{\mathscr{B}}(FA, -)$ is naturally equivalent to $\mathrm{Hom}_{\mathscr{A}}(A, U-)$ as a functor from $\mathscr{B}$ to* **Set**. *Then the definition of F on objects can be extended to arrows in such a way that F becomes a functor and is left adjoint to U.*

This theorem is called the **Pointwise Adjointness Theorem**. Its proof generalizes the argument of 9.1.3 but we omit the details. They can be found in [Barr and Wells, 1985], Section 1.9, Theorem 1, page 52. There is a detailed, general discussion of many equivalent definitions of adjunction in [Mac Lane, 1971], Chapter IV.

One application of this theorem is in showing that the definition of cartesian closed categories given in Chapter 7 is equivalent to the assumptions that the functors Δ of 9.2.3 and $- \times A$ of 9.2.5 have adjoints. In each case, the definition given in Chapter 7 can be input to the Pointwise Adjointness Theorem and the output is the adjoint described in the previous section.

This theorem has a simple formulation in terms of the universal elements of Section 5.7, as follows.

9.3.6 Proposition *A functor $U : \mathscr{B} \to \mathscr{A}$ has a left adjoint if and only if for each object A of $\mathscr{A}$, the functor $\mathrm{Hom}(A, U-) : \mathscr{B} \to$* **Set** *has a universal element.*

If $b : A \to UB$ is the universal element, then $FA = B$ and $b : A \to UB = UFA$ is the component at A of the natural transformation η that appears in 9.2.1.

As an example of how this proposition can be used, one can use it to deduce from Proposition 9.1.2 that $X \mapsto F(X)$ is the object map of a left adjoint to the underlying functor $U :$ **Mon** $\to$ **Set**. Of course, we proved this directly in 9.1.3.

9.3.7 Theorem *Let $F : \mathscr{A} \to \mathscr{B}$ be left adjoint to $U : \mathscr{B} \to \mathscr{A}$. Then U preserves limits and F preserves colimits.*

Proof. We sketch the proof that U preserves limits; the details are easy, but the notation is slightly unpleasant. Recall the definition of the cone functor $\mathrm{cone}(-, D) : \mathscr{C}^{\mathrm{op}} \to$ **Set** from 8.2.6. Then we have, for a diagram $D : \mathscr{I} \to \mathscr{B}$ with limit given by a cone $V \to D$, the following equivalences of contravariant functors:

$$\mathrm{cone}(-, UD) \cong \mathrm{cone}(F-, D) \cong \mathrm{Hom}(F-, V) \cong \mathrm{Hom}(-, UV)$$

which shows that UV is a limit of UD. The first isomorphism in that equation should be verified, since we extended the isomorphism of Theorem 9.3.2 from the hom functor to the cone functor, but that verification is straightforward. The second isomorphism follows from the fact that the cone $V \to D$ is a universal element of $\mathrm{cone}(-, D)$, so that $\mathrm{cone}(-, D) \cong \mathrm{Hom}(-, V)$. The third isomorphism follows from Theorem 9.3.2. □

The interesting question is the extent to which the converse is true. The basic fact is that the converse is false. First, the category $\mathscr{B}$ may not have enough limits for the condition to be meaningful. The really interesting case is the one in which every (small) diagram in $\mathscr{B}$ has a limit. Even in that case, there is still what can basically be described as a size problem. To go into this in more detail is beyond the scope of these notes. The best result is Freyd's Adjoint Functor Theorem. See [Barr and Wells, 1985], Section 1.9, Theorem 3, page 54 or [Mac Lane, 1971], Section V.6. Another reference for the Adjoint Functor Theorem (with a different point of view) is [Freyd and Scedrov, 1990], pages 144–146.

10. Triples

A triple is a structure which abstracts the idea of adjunction; part of the structure is an endofunctor. Triples have turned out to be an important technical tool in category theory. Section 10.1 defines triples and gives some of their basic properties. In Section 10.2 we develop the idea of an algebra for a triple; it is an algebra for the endofunctor part of the triple with certain properties.

10.1 Triples

We now describe a structure based on an endofunctor which has turned out to be an important technical tool in studying toposes and related topics. It is an abstraction of the concept of adjoint and in a sense an abstraction of universal algebra (see the remarks in fine print at the end of 10.2.3 below).

10.1.1 Algebras for an endofunctor Let $\mathscr{A}$ be a category and $R : \mathscr{A} \longrightarrow \mathscr{A}$ be an endofunctor. An **R-algebra** is a pair (A, a) where $a : RA \longrightarrow A$ is an arrow of $\mathscr{A}$. A homomorphism between R-algebras (A, a) and (B, b) is an arrow $f : A \longrightarrow B$ of $\mathscr{A}$ such that

$$\begin{array}{ccc} RA & \xrightarrow{a} & A \\ {\scriptstyle Rf}\downarrow & & \downarrow{\scriptstyle f} \\ RB & \xrightarrow[b]{} & B \end{array}$$

commutes. This construction gives a category $(R : \mathscr{A})$ of R-algebras.

10.1.2 Definition A **triple $\mathsf{T} = (T, \eta, \mu)$** on a category $\mathscr{A}$ consists of a functor $T : \mathscr{A} \longrightarrow \mathscr{A}$, together with two natural transformations $\eta : \mathrm{id} \longrightarrow T$ and $\mu : T^2 \longrightarrow T$ for which the following diagrams commute.

$$\begin{array}{ccccc} T & \xrightarrow{\eta T} & T^2 & \xleftarrow{T\eta} & T \\ & \searrow^{=} & \downarrow{\scriptstyle \mu} & \swarrow_{=} & \\ & & T & & \end{array} \qquad \begin{array}{ccc} T^3 & \xrightarrow{T\mu} & T^2 \\ {\scriptstyle \mu T}\downarrow & & \downarrow{\scriptstyle \mu} \\ T^2 & \xrightarrow[\mu]{} & T \end{array} \tag{10.1}$$

Here, ηT and $T\eta$ are defined as in 5.6.3 and 5.6.4.

The transformation η is the **unit** of the triple and μ is the **multiplication**. The left diagram constitutes the (left and right) **unitary identities** and the right one the **associative identity**. The reason for these names comes from the analogy between triples and monoids. This will be made clear in 10.1.5.

Another widely used name for triple is 'monad'. However, they have nothing to do with the monads of Robinson's theory of infinitesimals.

10.1.3 The triple arising from an adjoint pair An adjoint pair gives rise to a triple on the domain of the left adjoint.

10.1.4 Proposition *Let $U : \mathscr{B} \longrightarrow \mathscr{A}$ and $F : \mathscr{A} \longrightarrow \mathscr{B}$ be functors such that $F \dashv U$ with $\eta : \mathrm{id} \longrightarrow UF$ and $\epsilon : FU \longrightarrow \mathrm{id}$ the unit and counit, respectively. Then $(UF, \eta, U\epsilon F)$ is a triple on $\mathscr{A}$.*

Note that $U\epsilon F : UFUF \longrightarrow UF$, as required for the multiplication of a triple with functor UF.

Conversely, every triple arises in that way out of some (generally many) adjoint pair. See Section 10.2 for two ways of constructing such adjoints.

10.1.5 Representation triples Let M be a monoid. The representation triple $\mathbf{T} = (T, \eta, \mu)$ on the category of sets is given by letting $T(S) = M \times S$ for a set S. $\eta S : S \to T(S) = M \times S$ takes an element $s \in S$ to the pair $(1, s)$. We define μS by $(\mu S)(m_1, m_2, s) = (m_1 m_2, s)$ for $s \in S,\ m_1, m_2 \in M$. Thus the unit and multiplication of the triple arise directly from that of the monoid. The unitary and associativity equations can easily be shown to follow from the corresponding equations for the monoid.

The standard way of getting this triple from an adjoint pair is by using the underlying and free functors on M-sets (see 4.2.1). If S and T are M-sets, then a function $f : S \to T$ is said to be **M-equivariant** if $f(ms) = mf(s)$ for $m \in M,\ s \in S$. For a fixed monoid M, the M-sets and the M-equivariant functions form a category, called the category of M-sets.

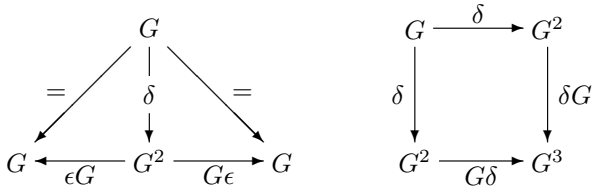
The **free M-set** generated by the set S is the set $M \times S$ with action given by $m'(m, s) = (m'm, s)$. Using Theorem 9.3.5, one can show immediately that this determines a functor left adjoint to the underlying set functor on the category of M-sets. The triple associated to this adjoint pair is the one described above.

10.1.6 The Kleene triple Let $T : \mathbf{Set} \to \mathbf{Set}$ be the functor which takes a set A to the Kleene closure A^* and a function $f : A \to B$ to the function $f^* : A^* \to B^*$ defined in Section 2.5.7. Let $\eta A : A \to A^*$ take an element a to the one-element string (a), and let $\mu A : A^{**} \to A^*$ take a string $(s_1, s_2, \ldots, s_k)$ of strings to the concatenated string $s_1 s_2 \cdots s_n$ in A^* obtained in effect by erasing inner brackets: thus $((a, b), (c, d, e), (), (a, a))$ goes to

$$(a, b)(c, d, e)()(a, a) = (a, b, c, d, e, a, a)$$

In particular, $\mu A((a, b)) = (a, b)$. Then $\eta : \mathrm{id} \to T$ and $\mu : T \circ T \to T$ are natural transformations, and (T, η, μ) is a triple.

10.1.7 Cotriples A **cotriple $\mathbf{G} = (G, \epsilon, \delta)$** in a category $\mathscr{A}$ is a triple in $\mathscr{A}^{\mathrm{op}}$. Thus G is an endofunctor of $\mathscr{A}$ and $\epsilon : G \to \mathrm{id}$ and $\delta : G \to G^2$ are natural transformations such that



(10.2)

Cotriples are used in Chapter 12.

10.2 Factorizations of a triple

10.2.1 The Kleisli category for a triple Let $\mathbf{T} = (T, \eta, \mu)$ be a triple on $\mathscr{C}$. We describe here a construction which exhibits the triple as coming from an adjoint. This construction, which is due to Kleisli [1965], has proven to be quite useful in theoretical computer science.

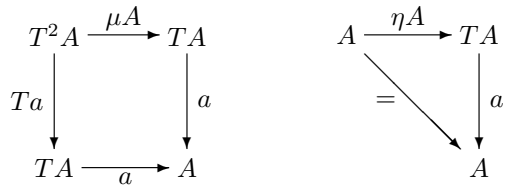
We define a category $\mathscr{K} = \mathscr{K}(\mathbf{T})$ which has the same objects as $\mathscr{C}$. If A and B are objects of $\mathscr{C}$, then an arrow in $\mathscr{K}$ from A to B is an arrow $A \to TB$ in $\mathscr{C}$. The composition of arrows is as follows. If $f : A \to TB$ and $g : B \to TC$ are arrows in $\mathscr{C}$, we let their composite in $\mathscr{K}$ be the following composite in $\mathscr{C}$:

$$A \xrightarrow{f} TB \xrightarrow{Tg} T^2C \xrightarrow{\mu C} TC$$

The identity of the object A is the arrow $\eta A : A \to TA$. It can be shown that this defines a category. Moreover, there are functors $U : \mathscr{K}(\mathbf{T}) \to \mathscr{C}$ and $F : \mathscr{C} \to \mathscr{K}(\mathbf{T})$ defined by $UA = TA$ and $Uf = \mu B \circ Tf$, where B is the codomain of f, and $FA = A$ and for $g : A \to B$, $Fg = Tg \circ \eta A$. Then F is left adjoint to U and $T = U \circ F$.

10.2.2 Eilenberg–Moore algebras Here is a second way, due to Eilenberg and Moore [1965] of factoring every triple as an adjoint pair of functors. In mathematics, this construction has been much more interesting than the Kleisli construction, but in computer science it has been quite the opposite.

Let $\mathbf{T} = (T, \eta, \mu)$ be a triple on $\mathscr{A}$. A T-algebra (A, a) is called a $\mathbf{T}$-algebra if the following two diagrams commute:



An arrow (homomorphism) between $\mathbf{T}$-algebras is the same as an arrow between the corresponding T-algebras. With these definitions, the $\mathbf{T}$-algebras form a category traditionally denoted $\mathscr{A}^{\mathbf{T}}$ and called the category of $\mathbf{T}$-algebras.

There is an obvious underlying functor $U : \mathscr{A}^{\mathbf{T}} \to \mathscr{A}$ with $U(A, a) = A$ and $Uf = f$. This latter makes sense because an arrow of $\mathscr{A}^{\mathbf{T}}$ *is* an arrow of $\mathscr{A}$ with special properties. There is also a functor $F : \mathscr{A} \to \mathscr{A}^{\mathbf{T}}$ given by $FA = (TA, \mu A)$ and $Ff = Tf$. Some details have to be checked; these are included in the following.

10.2.3 Proposition *The function F above is a functor left adjoint to U. The triple associated to the adjoint pair $F \dashv U$ is precisely $\mathbf{T}$.*

By a theorem of Linton's, every equationally defined category of one-sorted algebraic structures is in fact equivalent to the category of Eilenberg–Moore algebras for some triple in **Set** ([Barr and Wells, 1985], Theorem 5 of Section 4.3). In fact, the converse is true if infinitary operations are allowed (but then a hypothesis has to be added on the direct part of the theorem that there is only a set of operations of any given arity).

10.2.4 Example Let (T, η, u) be the triple in **Set** defined in 10.1.6. An algebra for this triple is a monoid: specifically, if $\alpha : T(A) \to A$ is an algebra, then the definition $ab = \alpha(a, b)$ makes A a monoid, and up to isomorphisms every monoid arises this way. Moreover, algebra homomorphisms are monoid homomorphisms, and every monoid homomorphism arises this way. The proof requires lengthy but not difficult verifications.

10.2.5 The Kleisli category and free algebras The Kleisli category of a triple $\mathbf{T} = (T, \eta, \mu)$ is equivalent to the full subcategory of free $\mathbf{T}$-algebras. Its definition makes it clear that the arrows are substitutions.

As an example, consider the list triple of 10.1.6. An arrow $f : A \to B$ (here A and B are sets) of the Kleisli category is a set function $A \to TB$, so that it associates a string of elements of B to each element of A. Suppose $A = \{a, b\}$ and $B = \{c, d, e\}$, and that $f(a) = cddc$ and $f(b) = ec$. Then $Tf : TA \to TTB$ takes, for example, the string $abba$ to $(cddc)(ec)(ec)(cddc)$, the result of substituting $cddc$ for a and ec for b in $abba$. Then μ takes that string to $cddcececcddc$. It is instructive in this situation to think of μ as *carrying out a computation*. In this case the computation is carried out using the (only) monoid operation, since in fact the algebras for this triple are monoids. Thus one can think of the objects of the Kleisli category as *computations*. This is more compelling if one uses a triple arising from algebraic structures such as rings that abstract some of the properties of numerical addition and multiplication; then the objects of the free algebra are polynomial expressions and μ evaluates the polynomial.

An important idea for developing this point of view is the notion of 'strong monad' [Kock, 1972], which has been developed by Moggi [1989, 1991b] and others [Cockett and Spencer, 1992], [Mulry, 1992]. Other applications of triples (monads) in computing science can be found in the survey [Rydeheard and Burstall, 1985], as well as [Moggi, 1991a], [Power, 1990], [Wadler, 1989], [Lüth and Ghani, 1997] and [Wadler, 1992] (the latter has many references to the literature).

Added reference to [Lüth and Ghani, 1997]

11. Toposes

A topos is a cartesian closed category with some extra structure which produces an object of subobjects for each object. This structure makes toposes more like the category of sets than cartesian closed categories generally are.

Toposes, and certain subcategories of toposes, have proved attractive for the purpose of modeling computation. A particular reason for this is that in a topos, a subobject of an object need not have a complement. One of the fundamental facts of computation is that it may be possible to list the elements of a subset effectively, but not the elements of its complement (see [Lewis and Papadimitriou, 1981], Theorems 6.1.3 and 6.1.4.). Sets which cannot be listed effectively do not exist for computational purposes, and toposes provide a universe of objects and functions which has many nice set-like properties but which does not force complements to exist. One specific subcategory of a topos, the category of modest sets, has been of particular interest in the semantics of programming languages (see [Barr and Wells, 1999], Chapter 15).

Toposes have interested mathematicians for other reasons. They are an abstraction of the concept of sheaf, which is important in pure mathematics. They allow the interpretation of second-order statements in the category in an extension of the language associated to cartesian closed categories in Chapter 7. This fact has resulted in toposes being proposed as an alternative to the category of sets for the foundations of mathematics. Toposes can also be interpreted as categories of sets with an internal system of truth values more general than the familiar two-valued system of classical logic; this allows an object in a topos to be thought of as a variable or time-dependent set, or as a set with various degrees of membership. In particular, most ways of defining the category of fuzzy sets lead to a category which can be embedded in a topos (see [Barr, 1986]).

Basic toposes are [Johnstone, 1977], [Barr and Wells, 1985], [Lambek and Scott, 1986], [Bell, 1988], [McLarty, 1992], [Mac Lane and Moerdijk, 1992]. None of these are aimed at applications to computer science. The most accessible introduction to the language and logic associated to a topos is perhaps that of [McLarty, 1992], Chapter 12. Other discussions of the language and the relation with logic are in [Makkai and Reyes, 1977], [Fourman, 1977], [Fourman and Vickers, 1986], [Boileau and Joyal, 1981]. The texts [Makkai and Reyes, 1977] and [Freyd and Scedrov, 1990] discuss toposes and also more general classes of categories that have a rich logical structure. The use of toposes specifically for semantics is discussed in [Hyland, 1982], [Hyland and Pitts, 1989], [Vickers, 1992].

11.1 Definition of topos

11.1.1 The subobject functor Recall from 3.3.12 that if C is an object of a category, a subobject of C is an equivalence class of monomorphisms $C_0 \rightarrowtail C$ where $f_0 : C_0 \rightarrowtail C$ is equivalent to $f_1 : C_1 \rightarrowtail C$ if and only if there are arrows (necessarily isomorphisms) $g : C_0 \to C_1$ and $h : C_1 \to C_0$ such that $f_1 \circ g = f_0$ and $f_0 \circ h = f_1$.

Assuming the ambient category $\mathscr{C}$ has pullbacks, the 'set of subobjects' function is the object function of a functor $\mathrm{Sub} : \mathscr{C}^{\mathrm{op}} \to \mathbf{Set}$: precisely, for an object C, $\mathrm{Sub}(C)$ is the set of subobjects of C. We must define Sub on arrows.

If $k : C' \to C$ is an arrow and if $f_0 : C_0 \rightarrowtail C$ represents a subobject of C, then in a pullback

$$\begin{array}{ccc} C'_0 & \xrightarrow{k_0} & C_0 \\ {\scriptstyle f'_0}\downarrow & & \downarrow{\scriptstyle f_0} \\ C' & \xrightarrow[k]{} & C \end{array} \tag{11.1}$$

the arrow f'_0 is also a monomorphism (see 8.3.4).

It is left as an exercise to prove, using the universal mapping property of pullbacks, that if the monomorphism $f_0 : C_0 \rightarrowtail C$ is equivalent to $f_1 : C_1 \rightarrowtail C$, then the pullbacks $f'_0 : C'_0 \rightarrowtail C'$ and $f'_1 : C'_1$

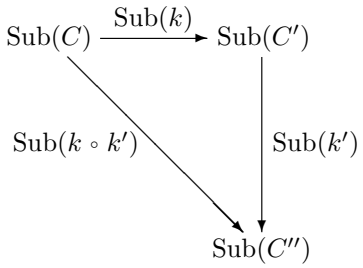
$\rightarrowtail C'$ are also equivalent. Thus not only is a pullback of a monomorphism a monomorphism, but also a pullback of a subobject is a subobject.

Thus we can define, for an arrow k as above,

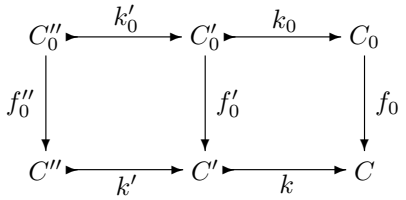
$$\mathrm{Sub}(k) : \mathrm{Sub}(C) \longrightarrow \mathrm{Sub}(C')$$

to be the function that sends the equivalence class containing f_0 to the equivalence class containing the pullback f_0'.

To show that this is a functor, we must show that the identity arrow induces the identity arrow on subobjects (exercise) and that if $k' : C'' \longrightarrow C'$, then the diagram



commutes. But the commutativity of this diagram at the subobject represented by f_0 is equivalent to the outer rectangle of the diagram



being a pullback when the two smaller squares are, which is easily verified.

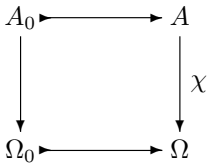
11.1.2 Definition A **topos** is a category which

TOP–1 has finite limits;

TOP–2 is cartesian closed;

TOP–3 has a representable subobject functor.

We know that a functor is representable if and only if it has a universal element (see 5.7.12). A universal element of the subobject functor is an object, usually called Ω, and a subobject $\Omega_0 \subseteq \Omega$ such that for any object A and subobject $A_0 \subseteq A$, there is a unique arrow $\chi : A \longrightarrow \Omega$ such that there is a pullback



It can be proved that Ω_0 is the terminal object and the left arrow is the unique arrow from A_0 ([Barr and Wells, 1985], Proposition 4 of Section 2.3).

The object Ω is called the **subobject classifier** and the arrow from $\Omega_0 = 1 \longrightarrow \Omega$ is usually denoted true. The arrow χ corresponding to a subobject is called the **characteristic arrow** of the subobject.

The fact that the subobject functor is represented by Ω means precisely that there is a natural isomorphism

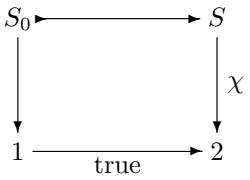
$$\phi : \mathrm{Sub}(-) \longrightarrow \mathrm{Hom}(-, \Omega)$$

which takes a subobject to its characteristic function.

11.1.3 Example The category of sets is a topos. It was shown in 7.1.9 that sets are a cartesian closed category. A two-element set, which we call 2, is a subobject classifier. In fact, call the two elements true and false. Given a set S and subset $S_0 \subseteq S$, define the characteristic function $\chi : S \longrightarrow \{\text{true}, \text{false}\}$ by

$$\chi(x) = \begin{cases} \text{true} & \text{if } x \in S_0 \\ \text{false} & \text{if } x \notin S_0 \end{cases}$$

Then the following square (where the top arrow is inclusion) is a pullback:



Thus 2 is a subobject classifier.

11.2 Properties of toposes

We list here some of the properties of toposes, without proof.

11.2.1 In the first place, a topos is not only cartesian closed, it is "locally cartesian closed" (meaning that every slice is cartesian closed — this is described in detail in [Barr and Wells, 1999], Section 13.4). This is Corollary 1.43, p. 36 of [Johnstone, 1977], Corollary 7, p. 182 of [Barr and Wells, 1985] or section 17.2 of [McLarty, 1992].

11.2.2 Power objects In any topos, the object $[A \longrightarrow \Omega]$ has the property that

$$\text{Hom}(B, [A \longrightarrow \Omega]) \cong \text{Hom}(A \times B, \Omega) \cong \text{Sub}(A \times B) \tag{11.2}$$

These isomorphisms are natural when the functors are regarded as functors of either A or of B. (One of them appears, for **Set**, in Example 5.3.8.) The object $[A \longrightarrow \Omega]$ is often called the **power object** of A and denoted $\mathscr{P}A$. It is the topos theoretic version of the powerset of a set. Theorem 1 of Section 5.4 of [Barr and Wells, 1985] implies that a category with finite limits is a topos if for each object A there is a power object that satisfies (11.2).

The inverse image and universal image constructions in 9.2.8 for the powerset of a set can be made on $[A \longrightarrow \Omega]$ for any object A in a topos. The left and right adjoints of the pullback functors (they exist because any topos is locally cartesian closed) are related to these images via the diagram in [Johnstone, 1977], Proposition 5.29; this diagram is called the 'doctrinal diagram' and is the basis for introducing elementary (first order) logic into a topos.

11.2.3 Effective equivalence relations Let $d, e : E \rightrightarrows A$ be two arrows in a category. For any object B we have a single function

$$\langle \text{Hom}(B, d), \text{Hom}(B, e) \rangle : \text{Hom}(B, E) \longrightarrow \text{Hom}(B, A) \times \text{Hom}(B, A)$$

which sends f to the pair $(d \circ f, e \circ f)$. If this function is, for each object B, an isomorphism of $\text{Hom}(B, E)$ with an equivalence relation on the set $\text{Hom}(B, A)$, then we say that E is an **equivalence relation** on the object A. This means that the image of $\langle \text{Hom}(B, d), \text{Hom}(B, e) \rangle$ is actually an equivalence relation on $\text{Hom}(B, A)$.

This can be thought of as embodying two separate conditions. In the first place, the function $\langle \text{Hom}(B, d), \text{Hom}(B, e) \rangle$ must be an injection, because we are supposing that it maps $\text{Hom}(B, E)$ isomorphically to a subset of $\text{Hom}(B, A) \times \text{Hom}(B, A)$. Secondly, that subset must satisfy the usual reflexivity, symmetry and transitivity conditions of equivalence relations.

11.2.4 Kernel pairs Here is one case in which this condition is automatic. If $g : A \to C$ is an arrow, the pullback of the square

$$\begin{array}{ccc} E & \xrightarrow{d} & A \\ {\scriptstyle e}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[g]{} & C \end{array}$$

is called a **kernel pair** of g. Notation: we will write that

$$E \underset{e}{\overset{d}{\rightrightarrows}} A \xrightarrow{g} C$$

is a kernel pair. For an object B of $\mathscr{C}$, the definition of this limit is that there is a one to one correspondence between arrows $B \to E$ and pairs of arrows (h, k) from B to A such that $g \circ h = g \circ k$ and that the correspondence is got by composing an arrow from B to E with d and e, resp. To put it in other words, $\mathrm{Hom}(B, E)$ is isomorphic to

$$\{(h, k) \in \mathrm{Hom}(B, A) \times \mathrm{Hom}(B, A) \mid g \circ h = g \circ k\}$$

which is an equivalence relation.

11.2.5 Suppose that $E \rightrightarrows A$ describes an equivalence relation. We say that the equivalence relation is **effective** if it is a kernel pair of some arrow from A. We say that a category has **effective equivalence relations** if every equivalence relation is effective. We give the following without proof. The interested reader may find the proof in [Barr and Wells, 1985] Theorem 7 of Section 2.3, [Johnstone, 1977], Proposition 1.23, p. 27, or [McLarty, 1992], Section 16.7.

11.2.6 Theorem *In a topos, every equivalence relation is effective.*

11.2.7 Example Equivalence relations in the categories **Set** and **Mon** are effective. An equivalence relation in **Set** is simply an equivalence relation, and the class map to the quotient set is a function that has the equivalence relation as kernel pair. An equivalence relation on a monoid M in **Mon** is a congruence relation on M; it is effective because a monoid multiplication can be defined on the quotient set of the congruence relation that makes the quotient map a homomorphism.

There are many categories which lack effective equivalence relations. One is the category of partially ordered sets and monotone maps. Here is a simple example. Let C be a two-element chain $x < y$. Consider the subset E of $C \times C$ consisting of all four pairs (x, x), (x, y), (y, x) and (y, y). The only ordering is that $(x, x) \leq (y, y)$. Then E is an equivalence relation, but is not the kernel pair of any arrow out of C. The least kernel pair that includes E has the same elements as E, but has the additional orderings $(x, x) \leq (x, y) \leq (y, y)$ and $(x, x) \leq (y, x) \leq (y, y)$.

Other important properties of toposes are contained in the following.

11.2.8 Theorem *Let $\mathscr{E}$ be a topos.*

(a) *$\mathscr{E}$ has finite colimits.*

(b) *$\mathscr{E}$ has finite disjoint and universal sums.*

(c) *Every epi in $\mathscr{E}$ is regular, and $\mathscr{E}$ is a regular category.*

An early proof of the fact that a topos has finite colimits ([Mikkelson, 1976]) mimicked the construction in sets. For example, the coequalizer of two arrows was constructed as a set of subsets, which can be identified as the set of equivalence classes mod the equivalence relation generated by the two arrows. However, the argument is difficult. The modern proof (due to Paré) is much easier, but it involves some legerdemain with triples and it is hard to see what it is actually doing. See [Barr and Wells, 1985], Corollary 2 of 5.1 for the latter proof. The rest is found in 5.5 of the same source.

11.2.9 The initial topos There is an finite-limit theory whose models are toposes. (See [Barr and Wells, 1985], Section 4.4. In that book, FL theories are called LE theories.) It follows that there is an initial model of the topos axioms. This topos lacks a natural numbers object. It might be an interesting model for a rigidly finitistic model of computation, but would not allow the modeling of such things as recursion.

The phrase 'initial topos' is usually reserved for the initial model of the axioms for toposes with a natural numbers object (See Section 5.5 of [Barr and Wells, 1999]). This category provides an interesting model for computation. The arrows from **N** to **N** in that category are, not surprisingly, the total recursive functions. In fact, all partial recursive functions are modeled in there as well, but as partial arrows, which we now describe.

11.2.10 Partial arrows In 2.1.13 we discussed partial functions between sets. This concept can be extended to any category. Let A and B be objects. A partial arrow A to B consists of a subobject $A_0 \subseteq A$ and an arrow $f : A_0 \longrightarrow B$. This defines the partial arrow f in terms of a particular representative $A_0 \longrightarrow A$ of the subobject, but given any other representative $A'_0 \longrightarrow A$, there is a unique arrow from A'_0 to A_0 commuting with the inclusions which determines an arrow from A'_0 to B by composition with f. The subobject determined by A_0 is called the **domain** of the partial arrow. If $g : A_1 \longrightarrow B$ is another partial arrow on A we say the $f \leq g$ if $A_0 \subseteq A_1$ and the restriction of g to A_0 is f. If we let $i : A_0 \longrightarrow A_1$ denote the inclusion arrow, then the second condition means simply that $g \circ i = f$. We will say that f and g are the same partial arrow if both $f \leq g$ and $g \leq f$. This means that the domains of f and g are the same subobject of A and that f and g are equal on that domain.

We say that **partial arrows to B are representable** if there is an object $\widetilde{B}$ and an embedding $B \rightarrowtail \widetilde{B}$ such that there is a one to one correspondence between arrows $A \longrightarrow \widetilde{B}$ and partial arrows A to B, the correspondence given by pulling back:

$$\begin{array}{ccc} A_0 & \longrightarrow & A \\ \downarrow & & \downarrow \\ B & \longrightarrow & \widetilde{B} \end{array}$$

In a topos, the arrow true: $1 \longrightarrow \Omega$ represents partial functions to 1. The reason is that since each object has a unique arrow to 1, a partial arrow from A to 1 is equivalent to a subobject of A.

11.2.11 Theorem *In a topos, partial arrows into any object are representable.*

See [Johnstone, 1977], Section 1.26 or [McLarty, 1992], Section 17.1 for the proof.

11.3 Presheaves

11.3.1 Definition Let $\mathscr{C}$ be a category. A functor $E : \mathscr{C}^{\mathrm{op}} \longrightarrow \mathbf{Set}$ is called a **presheaf** on $\mathscr{C}$. Thus a presheaf on $\mathscr{C}$ is a contravariant functor. The presheaves on $\mathscr{C}$ with natural transformations as arrows forms a category denoted $\mathbf{Psh}(\mathscr{C})$.

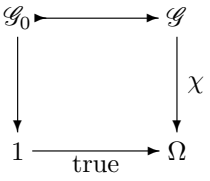
We considered presheaves as actions in Section 3.2. They occur in other guises in the categorical and computer science literature, too. For example, a functor $F : A \longrightarrow \mathbf{Set}$, where A is a set treated as a discrete category, is a 'bag' of elements of A. If $a \in A$, the set $F(a)$ denotes the multiplicity to which a occurs in A. See [Taylor, 1989] for an application in computing science.

11.3.2 Proposition *The category of presheaves on a category $\mathscr{C}$ form a topos.*

The proof may be found in [Barr and Wells, 1985] Section 2.1, Theorem 4. That proof uses a different, but equivalent, definition of topos.

11.3.3 Example Consider the category we will denote by $0 \rightrightarrows 1$. It has two objects and four arrows, two being the identities. A contravariant set-valued functor on this category is a pair of objects G_0 and G_1 and a pair of arrows we will call source, target : $G_1 \longrightarrow G_0$. The two identities are sent to identities. Thus the category of presheaves on this category is the category of graphs, which is thereby a topos.

We described the exponential object for graphs in 7.1.12. It is instructive to see what the subobject classifier is. We have to find a graph Ω and an arrow true : $1 \longrightarrow \Omega$ such that for any graph $\mathscr{G}$ and subgraph $\mathscr{G}_0 \subseteq \mathscr{G}$, there is a unique graph homomorphism $\chi : \mathscr{G} \longrightarrow \Omega$ such that the diagram



commutes.

We define the graph Ω as follows. It has five arrows we will call 'all', 'source', 'target', 'both' and 'neither'. The reason for these names should become clear shortly. It has two nodes we will call 'in' and 'out'. The arrows 'all' and 'both' go from 'in' to itself, 'neither' goes from 'out' to itself. The arrow 'source' goes from 'in' to 'out' and 'target' from 'out' to 'in'. The terminal graph, which has one arrow and one node, is embedded by the function true that takes the arrow to 'all' and the node to 'in'.

Now given a graph $\mathscr{G}$ and a subgraph $\mathscr{G}_0$ we define a function $\chi : \mathscr{G} \longrightarrow \Omega$ as follows. For a node n of $\mathscr{G}$, $\chi(n)$ is 'in' or 'out', according to whether n is in $\mathscr{G}_0$ or not. For an arrow a, we let $\chi(a)$ be 'all' if $a \in \mathscr{G}_0$ (whence its source and target are as well). If not, there are several possibilities. If the source but not the target of a belongs to $\mathscr{G}_0$, then $\chi(a) =$'source'. Similarly, if the target but not the source is in $\mathscr{G}_0$, it goes to 'target'. If both the source and target are in it, then $\chi(a) =$'both' and if neither is, then it goes to 'neither'.

11.4 Sheaves

The general definition of sheaves requires a structure on the category called a **Grothendieck topology**. The most accessible and detailed discussion of Grothendieck topologies is that of [Mac Lane and Moerdijk, 1992]. Here we will discuss the special case of sheaves in which the category is a partial order.

11.4.1 Let P be a partially ordered set. From the preceding section, a presheaf E on P assigns to each element $x \in P$ a set $E(x)$ and whenever $x \leq y$ assigns a function we will denote $E(x, y) : E(y) \longrightarrow E(x)$ (note the order; *x precedes y, but the arrow is from $E(y)$ to $E(x)$*). This is subject to two conditions. First, that $E(x, x)$ be the identity function on $E(x)$ and second that when $x \leq y \leq z$, that $E(x, y) \circ E(y, z) = E(x, z)$. The arrows $E(x, y)$ are called **restriction functions**.

11.4.2 Heyting algebras We make the following supposition about P.

HA–1 There is a top element, denoted 1, in P.

HA–2 Each pair of elements $x,\ y \in P$ has an infimum, denoted $x \wedge y$.

HA–3 Every subset $\{x_i\}$ of elements of P has a supremum, denoted $\bigvee x_i$.

HA–4 For every element $x \in P$ and every subset $\{x_i\} \subseteq P$, $x \wedge (\bigvee x_i) = \bigvee (x \wedge x_i)$.

A poset that satisfies these conditions is called a **complete Heyting algebra**. Complete Heyting algebras have an operation corresponding to implication: Let $\mathscr{H}$ be a complete Heyting algebra. Define the binary operation $\Rightarrow$: $\mathscr{H} \times \mathscr{H} \longrightarrow \mathscr{H}$ by requiring that $a \Rightarrow b$ is the join of all elements c for which $a \wedge c \leq b$. Then $a \wedge c \leq b$ if and only if $c \leq a \Rightarrow b$. When $\mathscr{H}$ is regarded as a category in the usual way, it is cartesian closed with $\Rightarrow$ as internal hom.

11.4.3 If $\{x_i\}$ is a subset with supremum x, and E is a presheaf, there is given a restriction function $e_i : E(x) \longrightarrow E(x_i)$ for each i. The universal property of product gives a unique function $e : E(x) \longrightarrow \prod_i E(x_i)$ such that $p_i \circ e = e_i$. In addition, for each pair of indices i and j, there are functions $c_{ij} : E(x_i) \longrightarrow E(x_i \wedge x_j)$ and $d_{ij} : E(x_j) \longrightarrow E(x_i \wedge x_j)$ induced by the relations $x_i \geq x_i \wedge x_j$ and $x_j \geq x_i \wedge x_j$. This gives two functions $c, d : \prod_i E(x_i) \longrightarrow \prod_{ij} E(x_i \wedge x_j)$ such that

$$\begin{array}{ccc} \prod_i E(x_i) & \xrightarrow{\ c\ } & \prod_{ij} E(x_i \wedge x_j) \\ \downarrow{\scriptstyle p_i} & & \downarrow{\scriptstyle p_{ij}} \\ E(x_i) & \xrightarrow[c_{ij}]{} & E(x_i \wedge x_j) \end{array}$$

and

$$\begin{array}{ccc} \prod_i E(x_i) & \xrightarrow{\ d\ } & \prod_{ij} E(x_i \wedge x_j) \\ \downarrow{\scriptstyle p_i} & & \downarrow{\scriptstyle p_{ij}} \\ E(x_i) & \xrightarrow[d_{ij}]{} & E(x_i \wedge x_j) \end{array}$$

commute.

11.4.4 Definition A presheaf is called a **sheaf** if it satisfies the following additional condition:

$$x = \bigvee x_i$$

implies

$$E(x) \xrightarrow{e} \prod_i E(x_i) \underset{d}{\overset{c}{\rightrightarrows}} \prod_{ij} E(x_i \wedge x_j)$$

is an equalizer.

11.4.5 Theorem *The category of sheaves on a Heyting algebra is a topos.*

As a matter of fact, the category of sheaves for any Grothendieck topology is a topos (see any of the texts [Johnstone, 1977], [Barr and Wells, 1985], [Mac Lane and Moerdijk, 1992], [McLarty, 1992]).

11.4.6 Constant sheaves A presheaf E is called **constant** if for all $x \in P$, $E(x)$ is always the same set and for all $x \leq y$, the function $E(y, x)$ is the identity function on that set.

The constant presheaf at a one-element set is always a sheaf. This is because the sheaf condition comes down to a diagram

$$1 \longrightarrow 1 \rightrightarrows 1$$

which is certainly an equalizer. No constant presheaf whose value is a set with other than one element can be a sheaf. In fact, the 0 (bottom) element of P is the supremum of the empty set and the product of the empty set of sets is a one-element set (see 6.3.6). Hence the sheaf condition on a presheaf E is that

$$E(0) \rightarrow \prod_{\emptyset} \rightrightarrows \prod_{\emptyset}$$

which is

$$E(0) \rightarrow 1 \rightrightarrows 1$$

and this is an equalizer if and only if $E(0) = 1$.

11.4.7 A presheaf is said to be **nearly constant** if whenever $0 < x \leq y$ in P, the restriction $E(y) \to E(x)$ is an isomorphism. It is interesting to inquire when a nearly constant presheaf is a sheaf. It turns out that every nearly constant presheaf over P is a sheaf over P if and only if the meet of two nonzero elements of P is nonzero.

To see this, suppose E is a nearly constant presheaf whose value at any $x \neq 0$ is S and that $x = \bigvee x_i$. In the diagram

$$E(x) \to \prod E(x_i) \rightrightarrows \prod E(x_i \wedge x_j)$$

every term in which $x_i = 0$ contributes nothing to the product since $1 \times Y \cong Y$. An element of the product is a string $\{s_i\}$ such that $s_i \in S$. The condition of being an element of the equalizer is the condition that the image of s_i under the induced function $E(x_i) \to E(x_i \wedge x_j)$ is the same as the image of s_j under $E(x_j) \to E(x_i \wedge x_j)$. But in a nearly constant sheaf, all these sets are the same and all the functions are the identity, so this condition is simply that $s_i = s_j$. But this means that an element of the equalizer must be the same in every coordinate, hence that diagram is an equalizer.

11.4.8 Interpretation of sheaves Let E be a sheaf on P. The reader will want to know how E is to be interpreted. Start by thinking of P as an algebra of truth values. Whereas propositions (assertions) in ordinary logic are either true or false (so that ordinary logic is often called 2-valued), in P-valued logic, the truth of an assertion is an element of P. In the next section, we will use this idea to discuss time sheaves in which propositions can be true at some times and not at others. If p is some proposition, let us write $[\![p]\!]$ to denote the element of P that is the truth value.

A sheaf E is a set in this logic. For $x \in P$, the (ordinary) set $E(x)$ could, as a first approximation, be thought of as the set of all entities a for which $[\![a \in E]\!]$ is at least x. If $y < x$, then $[\![a \in E]\!] \geq x$ implies that $[\![a \in E]\!] \geq y$ so that $E(x) \subseteq E(y)$. This is only a first approximation and what we have described is actually a P-valued fuzzy set (see Section 15.6 of [Barr and Wells, 1999]). The reason is that equality is also a predicate and it may happen, for example, that $[\![a = b]\!]$ could lie between x and y so that the entities a and b are not discernably equal at level x, but are equal at level y. The result is that rather than an inclusion, we have a restriction function $E(x) \to E(y)$ for $y \leq x$.

11.4.9 Time sheaves, I Here is a good example of a topos in which one can see that the restriction arrows should not be expected to be injective. Consider the partially ordered set whose elements are intervals on the real line with inclusion as ordering. It is helpful to think of these as time intervals.

Now consider any definition of a naive set. Some possible definitions will be time invariant, such as the set of mathematical statements about, say, natural numbers, that are true. Others of these 'sets' change with time; one example might be the set of all books that are currently in print; another the set of statements currently known to be true about the natural numbers. These may conveniently be thought of as the presheaves whose value on some time interval is the set of books that were in print over the entire interval and the set of statements about natural numbers known to be true during that entire interval. The restriction to a subinterval is simply the inclusion of the set of books in print over the whole interval to that (larger) set of those in print over that subinterval or the restriction of the knowledge from the interval to the subinterval. In this example, the restrictions are injective.

Instead of books in print, we could take the example of businesses in operation. Because of the possibility of merger, divestment and the like, two businesses which are actually distinct over a large interval might coincide over a smaller subinterval. Thus for this example of a "set", the restriction function is not injective in general.

Another situation in which the restriction functions are not necessarily injective arises from the set of variables in a block-structured programming language. The presheaf is this: the set for a certain time interval during the running of the program is the quotient of the set of variables which exist over the whole time interval, modulo the equivalence relation that identifies two variables in an interval if they should happen to have the same value over the whole interval. Two variables may not be equivalent over a large interval, whereas they may be equivalent over a smaller one; in that case the restriction function would not be injective.

In general, any property describes the set of all entities that have that property over the entire interval. The restriction to a subinterval arises out of the observation that any entity that possesses a property over an interval possesses it over any subinterval.

The sheaf condition in this case reduces to this: if the interval I is a union of subintervals I_k (where k ranges over an index set which need not be countable) and an entity possesses that property over every one of the subintervals I_k, then it does over the whole interval. This condition puts a definite restriction on the kinds of properties that can be used. For example, the property of being **grue**, that is blue over the first half of the interval and green on the second half, is not allowed. The properties that are allowed are what might be called **local**, that is true when they are true in each sufficiently small interval surrounding the point in time. This statement is essentially a tautology, but it does give an idea of what the sheaf condition means.

11.4.10 Time sheaves, II Here is another topos, rather different from the one above, that might also be considered to be time sheaves. Unlike the one above which is symmetric to forward and reverse time, this one definitely depends on which way time is flowing. This is not to say that one is better or worse, but they are different and have different purposes. In this one, the elements of the Heyting algebra are the times themselves. In other words, we are looking at time indexed sets, as opposed to sets indexed by time intervals.

We order this set by the reverse of the usual order. So a presheaf in this model is a family $\{X(t)\}$ of sets, t a real number, together with functions $f(s,t) : X(t) \longrightarrow X(s)$ for $t \leq s$, subject to the condition that $f(t,t)$ is the identity and $f(r,s) \circ f(s,t) = f(r,t)$ for $t \leq s \leq r$. The sheaf condition of Definition 11.4.4 is a bit technical, but can easily be understood in the special case of a presheaf all of whose transition arrows $f(s,t)$ are inclusions. In that case, the condition is that when $t = \bigwedge t_i$ (so that t is the greatest lower bound of the t_i), then $X(t) = \bigcap X(t_i)$.

An example, which might be thought typical, of such a sheaf might be described as the 'sheaf of states of knowledge'. At each time t we let $X(t)$ denote the set of all things known to the human race. On the hypothesis (which might be disputed) that knowledge is not lost, we have a function $X(t) \longrightarrow X(s)$ for $t \leq s$. In common parlance, we might consider this function to be an inclusion, or at least injective, but it is possible to modify it in various ways that will render it not so. We might introduce an equivalence relation on knowledge, so that two bits of knowledge might be considered the same. In that case, if at some future time we discover two bits of knowledge the same, then bits not equal at one time become equal at a later time and the transition arrow is not injective.

For example, consider our knowledge of the set of complex numbers. There was a time in our history when all the numbers e, i, π and -1 were all known, but it was not known that $e^{i\pi} = -1$. In that case, the number $e^{i\pi}$ and -1 were known separately, but not the fact that they were equal. See [Barr, McLarty and Wells, 1985]. The sheaf condition is this: if $\{t_i\}$ is a set of times and t is their infimum, then anything known at time t_i for every i is known at time t.

12. Categories with monoidal structure

Many of the categories that arise in computer science (and elsewhere) have a binary operation on objects and arrows. This operation is rarely actually associative, but is usually assumed associative up to natural isomorphism and subject to certain **coherence** laws as we will explain. We also suppose there is a unit object, which is not actually a unit, but is so up to a natural coherent isomorphism. Since the associativity and existence of a unit characterize monoids, these categories are called monoidal.

This chapter may be read immediately after Chapter 9.

12.1 Closed monoidal categories

12.1.1 Definition A **monoidal** category is a category $\mathscr{C}$ equipped with an object $\top$ and a functor $\otimes : \mathscr{C} \times \mathscr{C} \longrightarrow \mathscr{C}$. We use infix notation so that the value at (A, B) is written $A \otimes B$. A monoidal category must have, in addition, the natural isomorphisms a, r and l for all objects A, B and C listed in MC–1 through MC–3 below, and these must make Diagrams MC–5 and MC–6 commute. The category is a **symmetric monoidal category** if in addition it has the natural isomorphism SMC–4 and the diagrams SMC–7 through SMC–9 commute.

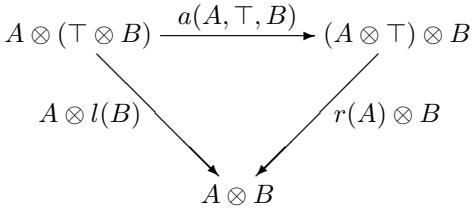
MC–1 $a(A, B, C) : A \otimes (B \otimes C) \longrightarrow (A \otimes B) \otimes C$;

MC–2 $rA : A \otimes \top \longrightarrow A$;

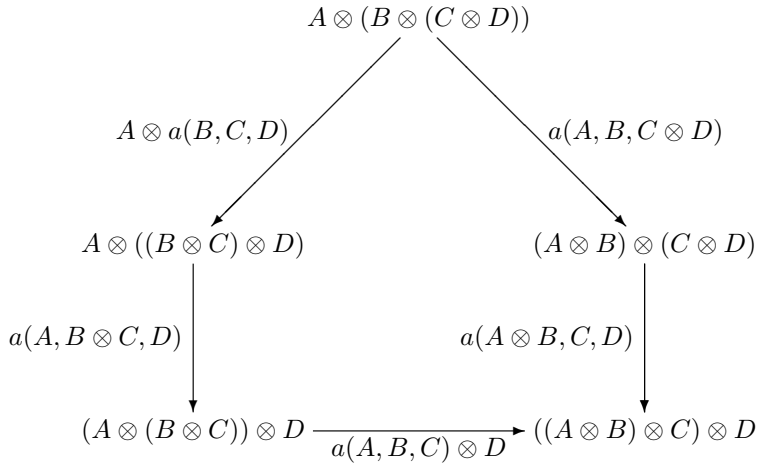
MC–3 $lA : \top \otimes A \longrightarrow A$.

SMC–4 $s(A, B) : A \otimes B \longrightarrow B \otimes A$.

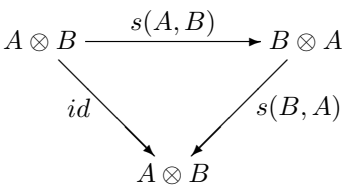
MC–5



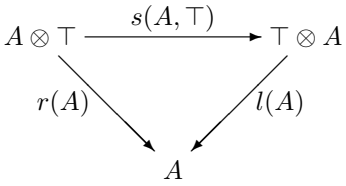
MC–6



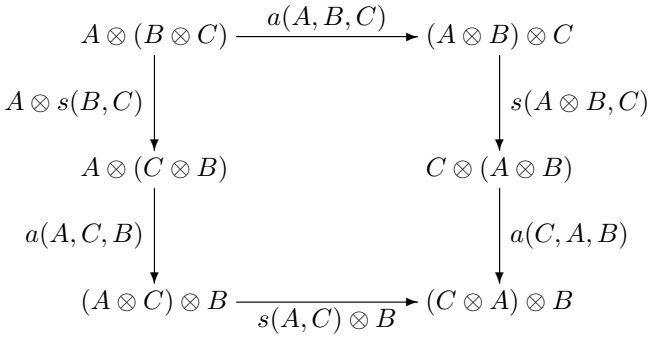
SMC–7



SMC–8



SMC–9



Each of the diagrams listed above asserts the commutativity of a diagram formed by composing isomorphisms built from instances of $\otimes$, a, r and l (and s in the symmetric case). These diagrams are chosen to be the axioms because assuming that they commute implies that *any* two parallel isomorphisms, provided that they are constructed from $\otimes$, a, r and l and s using $\otimes$ and composition, are the same. A precise statement of this fact is in Chapter VII.2 of [Mac Lane, 1971].

In any monoidal category $\mathscr{C}$, there is a functor $A \otimes - : \mathscr{C} \to \mathscr{C}$ that takes B to $A \otimes B$ and $f : B \to C$ to $A \otimes f : A \otimes B \to A \otimes C$.

12.1.2 Definition A monoidal category $\mathscr{C}$ is said to be **closed** if for each object A of $\mathscr{C}$, the functor $A \otimes -$ has a right adjoint.

If we denote the value at C of the right adjoint by $A \multimap C$ (A appears as a parameter since this is defined for each object A), then the defining condition is an isomorphism, natural in B and C,

$$\mathrm{Hom}(A \otimes B, C) \xrightarrow{\cong} \mathrm{Hom}(B, A \multimap C) \tag{12.1}$$

In the symmetric case, it follows that

$$\mathrm{Hom}(A \otimes B, C) \cong \mathrm{Hom}(A, B \multimap C)$$

In general, the latter isomorphism fails in the non-symmetric case; there we may want to suppose that there is, for each B a functor $C \mapsto C \circ\!\!- B$ such that $\mathrm{Hom}(A \otimes B, C) \cong \mathrm{Hom}(A, C \circ\!\!- B)$. We will not explore this point further, but will suppose from now on that our monoidal structure is symmetric.

12.1.3 Example A category that has finite products is automatically a symmetric monoidal category, using the categorical product for $\otimes$ and the terminal object for $\top$. The various isomorphisms can be constructed using the universal mapping property of products. For example, $a(A, B, C) : A \times (B \times C) \to (A \times B) \times C$ is the map that is denoted $\langle\langle \mathrm{proj}_1, \mathrm{proj}_1 \circ \mathrm{proj}_2\rangle, \mathrm{proj}_2 \circ \mathrm{proj}_2\rangle$ (see 6.2.9 for an explanation of this notation). The coherence is also automatic, essentially because of the uniqueness of the universal mappings. In Diagram MC–6, for example, both paths give the map

$$\langle\langle\langle p_1, p_1 \circ p_2\rangle, p_1 \circ p_2 \circ p_2\rangle, p_2 \circ p_2 \circ p_2\rangle : A \times (B \times (C \times D))$$
$$\to ((A \times B) \times C) \times D$$

where we have written p instead of proj.

Cartesian closed categories are, of course, just monoidal closed categories in which the product is the cartesian product. Thus any cartesian closed category is monoidal closed, but there are many others.

A category with finite sums is also monoidal, using the sum for $\otimes$ and the initial object for $\top$. The verification is essentially the same as for the case of products. **Set** (and many other categories with both products and sums) is therefore an example of a category with two inequivalent monoidal structures on it.

12.1.4 Example In 2.1.14, we constructed the category **Rel** of relations, whose objects are sets S, T, ..., and in which an arrow $S \to T$ is a subset $\alpha \subseteq S \times T$. The cartesian product of sets is not the product in this category – the disjoint sum is both sum and product – but it is a functor of two variables, that is a monoidal structure on **Rel**. We will denote it $S \otimes T$ to avoid confusion with the categorical product. An arrow $S \otimes T \to U$ in this category is a subset of $(S \otimes T) \otimes U \cong T \otimes (S \otimes U)$. Thus $\mathrm{Hom}(S \otimes T, U) \cong \mathrm{Hom}(T, S \otimes U)$. Thus **Rel** is a closed monoidal category with $S \multimap T = S \otimes T$.

Observe that this is an example of a closed monoidal category in which $A \multimap B$ is not a structure built on $\mathrm{Hom}(A, B)$: $A \multimap B = A \otimes B$ is the cartesian product (in **Set**, not **Rel**!) of A and B, whereas $\mathrm{Hom}(A, B)$ is the powerset of that cartesian product.

We return to this example in Example 12.3.3.

12.1.5 Subsets of a monoid Here is an example that is a poset viewed as a category. Let M be a monoid and $\mathscr{C}$ denote the category whose objects are the subsets of M with inclusions as the only morphisms. If A and B are subsets of M, let $A \otimes B = \{ab \mid a \in A \text{ and } b \in B\}$. Define $A \multimap C = \{b \in M \mid ab \in C \text{ for all } a \in A\}$. Then it is easy to verify that $A \otimes B \subseteq C$ if and only if $B \subseteq A \multimap C$. If we define $C \circ\!\!- B = \{a \in M \mid ab \in C \text{ for all } b \in B\}$, it is equally easy to see that $A \otimes B \subseteq C$ if and only if $A \subseteq C \circ\!\!- B$.

12.1.6 Sup semilattices Another example is the category of sup semilattices, (see 6.4.6), which also illustrates another principle, that it is often much easier to give an explicit description of the $\multimap$ operation than of the $\otimes$. If K and L are sup semilattices, then $K \multimap L$ is defined to be the set of all sup-semilattice homomorphisms from K to L. If f and g are such homomorphisms, then their sup is defined by $(f \vee g)(x) = f(x) \vee g(x)$. The bottom element is the constant function at the bottom element of L. It is easily seen that with these definitions $K \multimap L$ is a sup semilattice.

To get the monoidal structure, we will construct the left adjoint to the functor $K \multimap -$. The existence of the left adjoint also follows from the adjoint functor theorem (references to this are given after Theorem 9.3.7).

First, form the free sup semilattice generated by $K \times L$. It has elements of the form $(x_1, y_1) \vee (x_2, y_2) \vee \cdots \vee (x_n, y_n)$ with $x_1, x_2, \ldots, x_n \in K$ and $y_1, y_2, \ldots, y_n \in L$. These specifically include the empty sup 0. We then form the quotient semilattice gotten by all identifications of the form $(x_1, y) \vee (x_2, y) = (x_1 \vee x_2, y)$ and $(x, y_1) \vee (x, y_2) = (x, y_1 \vee y_2)$. In addition, we must identify the bottom element of the semilattice, which is the empty sup, with $(x, 0)$ for any $x \in K$ and with $(0, y)$ for any $y \in L$ These identifications have to be made compatible with the semilattice structure so that whenever two elements are identified, their joins with any third element are also identified.

It is instructive to see why any $(x, 0)$ has to be identified with 0. First, let us denote the image of (x, y) in $K \otimes L$ by $x \otimes y$. Not every element of $K \otimes L$ has the form $x \otimes y$, but it is generated by such elements. The isomorphism $\mathrm{Hom}(K \otimes L, M) \cong \mathrm{Hom}(L, K \multimap M)$ can be described as follows. If $f : K \otimes L \to M$ corresponds to $\widetilde{f} : L \to K \multimap M$, then for $y \in L$, $\widetilde{f}(y) : K \to M$ is defined by $\widetilde{f}(y)(x) = f(x \otimes y)$. In particular, $\widetilde{f}(0)$ is required to be the 0 homomorphism, in order to be a homomorphism of sup semilattices. Thus $f(x \otimes 0) = \widetilde{f}(0)(x) = 0$. In particular, take $M = K \otimes L$ and choose $f = \mathrm{id}$ to conclude that $x \otimes 0 = 0$. A similar argument will show that $0 \otimes y = 0$ as well, but in fact, it is not hard to show symmetry of $\otimes$ directly.

> The crucial property of sup semilattices exploited here is that sup semilattices are determined by operations, $\vee$ and 0 that are homomorphisms of the sup semilattice structure. Thus when L is a sup semilattice, $\vee : L \times L \to L$ is not just a function but a semilattice homomorphism. It is fairly uncommon for an equational theory to have the property that its operations are homomorphisms of the theory, but when it does, constructions analogous to the above will always give a monoidal closed structure.

Many other examples of closed monoidal categories are given in [Mac Lane, 1971]. Applications are discussed in [Asperti and Longo, 1991], [Corradini and Asperti, 1993], [Marti Oliet and Meseguer, 1991] and [Bartha, 1992].

12.2 Properties of $A \multimap C$

Monoidal closed categories exemplify one of the earliest perceptions of category theory, that a category is quite special if the set of arrows between any two objects is, in some natural way, also an object of the category. Thus, although it is much commoner to have some kind (or many kinds) of monoidal structure on a category than to have a closed structure, it is usually easier to define the closed structure, when it exists. This is well illustrated in the example of semilattices given in 12.1.6.

12.2.1 The internal hom Another aspect of this perception is that $A \multimap C$ is called the **internal hom** of A to C and indeed it has properties similar to those of the 'external' (that is the set-valued) hom functor $\mathrm{Hom}(A, C)$. For example, for each C, $A \mapsto A \multimap C$ is also functorial in A, albeit contravariantly. (For each A, $C \mapsto A \multimap C$ is a functor in C by definition.) This means that there is a natural way of defining, for each map $f : A' \to A$ a map $f \multimap C : A \multimap C \to A' \multimap C$ in such a way that the isomorphism (12.1) is natural in all three arguments. This generalizes Proposition 7.2.1.

Another aspect is illustrated by the following, which internalizes the adjunction. It is a generalization of isomorphism (7.1) and Proposition 7.2.2.

12.2.2 Proposition *In any monoidal closed category, there is a natural equivalence (of functors of three variables)*

$$(A \otimes B) \multimap C \cong B \multimap (A \multimap C)$$

Proof. The proof uses the Yoneda Lemma (Theorem 5.7.6 of Chapter 5) and the associativity isomorphism. At this point, we confine ourselves to constructing the isomorphism. We have, for any object D,

$$\begin{aligned} \mathrm{Hom}(D, (A \otimes B) \multimap C) &\cong \mathrm{Hom}((A \otimes B) \otimes D, C) \\ &\cong \mathrm{Hom}(A \otimes (B \otimes D), C) \\ &\cong \mathrm{Hom}(B \otimes D, A \multimap C) \\ &\cong \mathrm{Hom}(D, B \multimap (A \multimap C)) \end{aligned}$$

Each of these isomorphisms is easily seen to be natural in D and thus we conclude from the Yoneda Lemma that $(A \otimes B) \multimap C \cong A \multimap (B \multimap C)$. □

12.2.3 Example Let M be a commutative monoid. Recall from 4.2.1 that an M-action is a set S together with a function $\alpha : M \times S \to S$ subject to certain conditions. We generally denote $\alpha(m, s)$ by ms and never mention α explicitly. Suppose from now on that M is commutative. If S and T are M-actions, then there are two functions $M \times S \times T \to S \times T$, one taking (m, s, t) to (ms, t) and the other taking that element to (s, mt). The coequalizer of these two maps is denoted $S \otimes_M T$ or simply $S \otimes T$ if M is understood fixed. If we denote the class containing (s, t) by $s \otimes t$, then $S \otimes T$ can be thought of as consisting of elements $s \otimes t$, $s \in S$ and $t \in T$, subject to the relation that $ms \otimes t = s \otimes mt$ for $m \in M$. It is left as an exercise to show that this gives a monoidal structure on M-act with unit action the multiplication of M on itself. It is also a closed structure, given by defining $S \multimap T$ to be the set of equivariant maps S to T with mf defined by $(mf)(s) = m(f(s)) = f(ms)$, the latter equality from the fact that f is equivariant.

12.2.4 Evaluation and composition There are certain derived morphisms in a monoidal closed category, using the adjunction. For any objects A and B, the identity $A \multimap B \to A \multimap B$ corresponds to an arrow $e(A, B) : A \otimes (A \multimap B) \to A$ called **evaluation** since it internalizes the evaluation map. This is natural in A and B. We can then derive an arrow $c(A, B, C)$ defined as the map $(A \multimap B) \otimes (B \multimap C) \to A \multimap C$ that corresponds under the adjointness to the map

$$A \otimes (A \multimap B) \otimes (B \multimap C) \xrightarrow{e(A,B) \otimes \mathrm{id}} B \otimes (B \multimap C) \xrightarrow{e(B,C)} C$$

The arrow $c(A, B, C)$, which is also natural in all three variables, internalizes the composition arrow.

12.2.5 Cotriples in closed monoidal categories Let $\mathbf{G} = (G, \epsilon, \delta)$ be a cotriple on $\mathscr{C}$. The Kleisli category (often called the co-Kleisli category) $\mathscr{K} = \mathscr{K}(\mathbf{G})$ has the same objects as $\mathscr{C}$ and the hom sets are defined by $\mathscr{K}(A, B) = \mathscr{C}(GA, B)$. The composite of $f : GA \longrightarrow B$ and $g : GB \longrightarrow C$ is $g \circ Gf \circ \delta A$. This is the dual construction to the Kleisli category of a triple and the verifications are the same.

Note that we are using the name of the category to denote the hom sets. This notation is especially useful when we are dealing with two categories that have the same objects.

One of the most important properties of the Kleisli category of a cotriple is the following. It is used to construct models of classical logic inside linear logic.

12.2.6 Theorem *Suppose $\mathbf{G} = (G, \epsilon, \delta)$ is a cotriple on a symmetric monoidal closed category $\mathscr{C}$ with the property that for all objects A and B of $\mathscr{C}$, there are natural isomorphisms between the two variable functors $G(- \times -) \cong G - \otimes G -$. Then the Kleisli category for the cotriple is cartesian closed.*

Proof. Let $\mathscr{K}$ denote the Kleisli category. Then we have, for any objects A, B and C of $\mathscr{C}$ (and therefore of $\mathscr{K}$),

$$\mathscr{K}(A, B \times C) \cong \mathscr{C}(GA, B \times C) \cong \mathscr{C}(GA, B) \times \mathscr{C}(GA, C)$$

$$\cong \mathscr{K}(A, B) \times \mathscr{K}(A, C)$$

This isomorphism is clearly natural in A, which shows that, in $\mathscr{K}$, $B \times C$ is a product of B and C. (The same argument would work for any limit.) Then we have, again for any objects A, B and C of $\mathscr{C}$,

$$\mathscr{K}(A \times B, C) \cong \mathscr{C}(G(A \times B), C) \cong \mathscr{C}(GA \otimes GB, C)$$

$$\cong \mathscr{C}(GB, GA \multimap C) \cong \mathscr{K}(B, GA \multimap C)$$

so that the functor $GA \multimap -$ is right adjoint to $A \times -$ in $\mathscr{K}$. □

12.3 ∗-autonomous categories

12.3.1 Definition Let $\mathscr{C}$ be a symmetric monoidal closed category. A functor $(-)^* : \mathscr{C}^{\mathrm{op}} \longrightarrow \mathscr{C}$ is a **duality functor** if there is an isomorphism $d(A, B) : A \multimap B \xrightarrow{\cong} B^* \multimap A^*$, natural in A and B, such that for all objects A, B and C,

$$\begin{array}{ccc} (A \multimap B) \otimes (B \multimap C) & \xrightarrow{\quad c(A,B,C) \quad} & A \multimap C \\ \Big\downarrow{\scriptstyle d(A,B) \otimes d(B,C)} & & \Big\downarrow{\scriptstyle d(A,C)} \\ (B^* \multimap A^*) \otimes (C^* \multimap B^*) & \xrightarrow[\quad c(C^*, B^*, A^*) \circ s \quad]{} & C^* \multimap A^* \end{array}$$

commutes. In the bottom arrow, $s = s(B^* \multimap A^*, C^* \multimap B^*)$. A **∗-autonomous** category is a symmetric monoidal closed category with a given duality functor.

The discussion of ∗-autonomous categories in this section omits many details. The basic theory is in [Barr, 1979]. More recent developments are given in [Barr, 1996a] and [Barr, 1995]. ∗-autonomous categories form a model of linear logic [Barr, 1991], which has become important in modeling of parallel processes and is closely related to Petri nets and event structures. A good introduction, with references, to linear logic, is Yves Lafont's Appendix B of [Girard, Taylor and Lafont, 1989]. The connection between ∗-autonomous categories and linear logic is described in [Asperti and Longo, 1991], sections 4.4 and 5.5, and ∗-autonomous categories are used in modelling processes in [Pavlović and Abramsky, 1997].

Reference to [Pavlović and Abramsky, 1997] added

12.3.2 Properties of $*$-autonomous categories There is much redundancy in the data specifying a $*$-autonomous category. Given the duality, it is not hard to show that the $\otimes$ and $\multimap$ are related by the equations:

$$A \otimes B \cong (A \multimap B^*)^* \qquad A \multimap B \cong (A \otimes B^*)^*$$

so that only the $(-)^*$ and either $\multimap$ or $\otimes$ need be given explicitly. Of course, various isomorphisms, maps and coherences must still be given. The details can be found (for the not necessarily symmetric case) in [Barr, 1995]. It turns out that it is usually most convenient to describe the $\multimap$, since when the objects are structured sets, $A \multimap B$ usually turns out to be the set of structure preserving functions of $A \longrightarrow B$, equipped with a certain structure itself.

There is a second monoidal structure in a $*$-autonomous category, which we will denote by $\oplus$, defined by $A \oplus B = (A^* \otimes B^*)^*$. The unit for this monoidal structure is $\top^*$, which we will denote $\bot$, since $A \oplus \bot = (A^* \otimes \bot^*)^* \cong (A^* \otimes \top)^* \cong A^{**} \cong A$. Since $A^* \oplus C \cong A \multimap C$, there is a one-one correspondence between arrows $A \otimes B \longrightarrow C$ and arrows $B \longrightarrow A^* \oplus C$. This should be thought of as being analogous to the equivalence between the logical statements $a \wedge b \Rightarrow c$ and $b \Rightarrow \neg a \vee c$.

It is also possible to describe a $*$-autonomous category in terms of a symmetric closed monoidal category with an object $\bot$ subject to the condition that for every A, the map $A \longrightarrow ((A \multimap \bot) \multimap \bot)$ that corresponds under the adjunction to

$$(A \multimap \bot) \otimes A \xrightarrow{s(A \multimap \bot, A)} A \otimes (A \multimap \bot) \xrightarrow{e(A, \bot)} \bot$$

is an isomorphism. The reason is that we have

$$A^* \cong \top \multimap A^* \cong A \multimap \top^* \cong A \multimap \bot$$

which shows that the duality is determined by $\multimap$ and $\bot$. We say that $\bot$ is the **dualizing object** for the duality. It is clearly determined up to isomorphism as the dual of $\top$.

However, one of the most important ways (for computing science) of constructing $*$-autonomous categories does not work this way in that the $\multimap$ and $\otimes$ are constructed together. That is the so-called Chu construction, described in 12.5 below. We first present a familiar example.

12.3.3 Example The category **Rel** of sets and relations is a $*$-autonomous category. We saw that it was closed monoidal in Example 12.1.4. Since $\mathrm{Hom}(S,T) \cong \mathrm{Hom}(T,S)$ (both being the powerset of the cartesian product of S and T), it follows that we have a duality by letting $S^* = S$. This gives a $*$-autonomous category. In fact, it is an example of what is called a **compact** $*$-autonomous category, that is one in which $A \multimap B \cong A^* \otimes B$.

12.4 Factorization systems

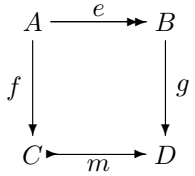
It is a familiar fact that every function in the category of sets can be factored as an epimorphism (surjection) followed by a monomorphism (injection). Similarly, every homomorphism in the category of abelian groups can be factored as an epimorphism followed by a monomorphism. The properties of these factorizations were abstracted early in the days of category theory, where they were known as bicategory structures. Under the name factorization system, they have a number of uses in category theory. We define them here because they are needed to state the main theorem on the Chu construction (Theorem 12.5.2).

12.4.1 Definition A **factorization system** in a category $\mathscr{C}$ consists of two subclasses $\mathscr{E}$ and $\mathscr{M}$ of the arrows of $\mathscr{C}$ subject to the conditions

FS—-1 If $\mathscr{I}$ is the class of isomorphisms, then $\mathscr{M} \circ \mathscr{I} \subseteq \mathscr{M}$ and $\mathscr{I} \circ \mathscr{E} \subseteq \mathscr{E}$.

FS—-2 Every arrow f in $\mathscr{C}$ factors as $f = m \circ e$ with $m \in \mathscr{M}$ and $e \in \mathscr{E}$.

FS—-3 In any commutative square



with $e \in \mathscr{E}$ and $m \in \mathscr{M}$, there is a unique $h : B \longrightarrow C$ such that $h \circ e = f$ and $m \circ h = g$.

The last condition is referred to as the "diagonal fill-in". If (as is the case in many examples) either every arrow in $\mathscr{M}$ is monic or every arrow in $\mathscr{E}$ is epic, then the uniqueness requirement in this condition may be omitted. Removed "Observe that"

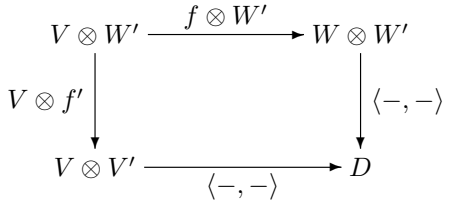
12.4.2 Example In **Set**, the class $\mathscr{E}$ of epimorphisms and the class $\mathscr{M}$ of monomorphisms constitute a factorization system. In many categories of algebraic structures (including monoids), the class $\mathscr{E}$ of regular epimorphisms (defined in Section 8.4.3) and the class $\mathscr{M}$ of monomorphisms constitute a factorization system. In the category of monoids and monoid homomorphisms, the class of all epis and all monos is not a factorization system.

The theory of factorization systems is developed in [Barr and Wells, 1999] and in [Borceux, 1994].

12.5 The Chu construction

Chu [1978, 1979] describes a construction that begins with a symmetric monoidal closed category and an object in it to use as dualizing object. He constructs from this a $*$-autonomous category that is closely related to the original category and has the chosen object as dualizing object. There is a non-symmetric version of this construction, but it is much more complicated [Barr, 1995], see also [Barr, 1996b].

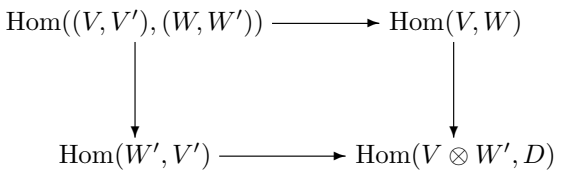
Suppose that $\mathscr{V}$ is a symmetric monoidal closed category and let D denote a fixed object of $\mathscr{V}$. We define a category $\mathscr{A}$ that has as objects triples $(V, V', \langle -, - \rangle)$ where V and V' are objects of $\mathscr{V}$ and $\langle -, - \rangle : V \otimes V' \longrightarrow D$ is an arrow of $\mathscr{V}$. We call $\langle -, - \rangle$ a pairing and we usually omit it in the notation, writing (V, V') for an object of $\mathscr{A}$. This notation means we have in mind a pairing from $V \otimes V'$ to D, and moreover this pairing will always be denoted $\langle -, - \rangle$. An arrow $(f, f') : (V, V') \longrightarrow (W, W')$ of $\mathscr{A}$ consists of a pair of arrows $f : V \longrightarrow W$ and $f' : W' \longrightarrow V'$ (note the direction of the second arrow) such that



commutes. This says that the arrows $(V, V') \longrightarrow (W, W')$ consist of all pairs

$$(f, f') \in \mathrm{Hom}(V, V') \times \mathrm{Hom}(W', W)$$

that give the same element of $\mathrm{Hom}(V \otimes W', D)$ or that



is a pullback. This observation is the key to the construction of the monoidal closed structure in $\mathscr{A}$.

The next step in the construction is to make the set of arrows into an object of $\mathscr{V}$. We define $\mathscr{V}((V,V'),(W,W'))$ so that

$$\begin{array}{ccc} \mathscr{V}((V,V'),(W,W')) & \longrightarrow & V \multimap W \\ \downarrow & & \downarrow \\ W' \multimap V' & \longrightarrow & (V \otimes W') \multimap D \end{array}$$

is a pullback. Here the arrow on the right side is described as follows. The arrow $\langle -,- \rangle : W \otimes W' \to D$ corresponds to an arrow $W \to W' \multimap D$ (the symmetry is involved here too) and then we have a composite arrow

$$V \multimap W \to V \multimap (W' \multimap D) \xrightarrow{\cong} (V \otimes W') \multimap D$$

The lower arrow in the diagram is similar.

If (V,V') is an object of $\mathscr{A}$, then so is (V',V) using $V' \otimes V \xrightarrow{s(V',V)} V \otimes V' \xrightarrow{\langle -,- \rangle} D$. We denote it by $(V,V')^*$. It is clear that

$$\mathrm{Hom}((V,V'),(W,W')) \cong \mathrm{Hom}((W,W')^*,(V,V')^*)$$

so that $(-)^*$ is a duality on $\mathscr{A}$. Now we can give the main definitions. We let

$$(V,V') \multimap (W,W') = (\mathscr{V}((V,V'),(W,W')), V \otimes W')$$

$$(V,V') \otimes (W,W') = (V \otimes W, \mathscr{V}((V,V'),(W,W')^*))$$

12.5.1 Separated and extensional Chu objects Suppose that $\mathscr{E}/\mathscr{M}$ is a factorization system on $\mathscr{V}$. Then we say that the Chu object (V,V') is **$\mathscr{M}$-separated** if the arrow $V \to V' \multimap D$ belongs to $\mathscr{M}$ and is **$\mathscr{M}$-extensional** if the arrow $V' \to V \multimap D$ belongs to $\mathscr{M}$. When the $\mathscr{M}$ remains fixed, we often omit it and speak of separated and extensional objects.

The reason for these names is this. We often think of a pair (V,V') as consisting of an object and a set of descriptions of scalar-valued functions on it. Separability is the existence of enough such descriptions in V' to distinguish (or separate) the elements of V, while extensionality is the word used to name the property of functions that two are equal if their values on all elements of their domain are equal. Functions have this property, but programs for computing them do not necessarily.

We denote the full subcategories of separated objects by $\mathrm{Chu}_s = \mathrm{Chu}_s(\mathscr{V},D)$ and of extensional objects by $\mathrm{Chu}_e = \mathrm{Chu}_e(\mathscr{V},D)$. We follow Vaughan Pratt in denoting the full subcategory of objects that are both separated and extensional by $\mathrm{chu} = \mathrm{chu}(\mathscr{V},D)$. Since Chu_s is evidently the dual of Chu_e, it is immediate that chu is self-dual. It is useful to ask if chu is also $*$-autonomous.

We also consider the following two conditions on factorization systems.

FC–1 Every arrow in $\mathscr{E}$ is an epimorphism;

FC–2 if $m \in \mathscr{M}$, then for any object A of $\mathscr{V}$, the induced $A \multimap m$ is in $\mathscr{M}$.

The following theorem gives conditions under which the separated, extensional objects in $\mathrm{Chu}(\mathscr{V},D)$ form a *-autonomous category.

12.5.2 Theorem *Suppose the category $\mathscr{V}$ has pullbacks and $\mathscr{E}/\mathscr{M}$ is a factorization system that satisfies FC–1 and FC–2. Then for any object D, the category* $\mathrm{chu}(\mathscr{V},D)$ *of $\mathscr{M}$-separated, and $\mathscr{M}$-extensional objects of* Chu *is $*$-autonomous.*

The proof is given in [Barr and Wells, 1999], Chapter 16.

12.5.3 Examples The first example is Chu(**Set**, 2). An object is simply a pair (S, S') of sets, together with a function $S \times S' \longrightarrow 2$. Equivalently, it is a subset of $S \times S'$, otherwise known as a relation between S and S'. The extensional objects are those for which S' is, up to isomorphism, a set of subsets of S and then $\langle s, s' \rangle = 1$ if $s \in s'$ and 0 otherwise. The separated ones are those for which, given $s_1,\ s_2 \in S$, there is at least one $s' \in S'$ with either $s_1 \in s'$ and $s_2 \notin s'$ or vice versa. A set together with a set of subsets is the beginnings of a topological space and the separation condition is the same as that of a T_1 topological space. This example has been intensively studied by Vaughan Pratt and his students.

A second example is given by taking $\mathscr{V}$ to be the category of vector spaces over a field F. The category of finite dimensional vector spaces is already a $*$-autonomous category with the set of linear transformations between two spaces as internal hom and the usual vector space dual as the duality operator. It is easily seen to be a $*$-autonomous subcategory of $\mathrm{Chu}(\mathscr{V}, F)$. It is also a $*$-autonomous subcategory of the separated extensional subcategory $\mathrm{chu}(\mathscr{V}, F)$. There is one distinctive feature of the separated extensional subcategory worth noting. In that case (and so far as is known only in that case), the tensor product and internal hom of separated extensional objects is already separated and extensional and there is no reason to apply the reflector and coreflector, respectively (see [Barr, 1996c]).

Bibliography

At the end of each entry, the pages on which that entry is cited are listed in parentheses.

Adámek, J. and J. Rosičky (1994). *Locally Presentable and Accessible Categories*. Cambridge University Press.

Asperti, A. and G. Longo (1991). *Categories, Types and Structures*. The MIT Press.

Backhouse, R., M. Bijsterveld, R. van Geldrop, and J. van der Woude (1995). 'Categorical fixed point calculus'. In Pitt et al. [1995], pages 159–179.

Backus, J. (1981). 'The algebra of functional programs: Function level reasoning, linear equations, and extended definitions'. In *Formalization of Programming Concepts*, J. Diaz and I. Ramos, editors, volume 107 of *Lecture Notes in Computer Science*. Springer-Verlag.

Backus, J. (1981). 'Is computer science based on the wrong fundamental concept of program?'. In *Algorithmic Languages*, J. W. deBakker and J. C. van Vliet, editors. North-Holland.

Barendregt, H. (1984). *The Lambda Calculus – its Syntax and Semantics*. North-Holland.

Barr, M. (1979). **-Autonomous Categories*, volume 752 of *Lecture Notes in Mathematics*. Springer-Verlag.

Barr, M. (1986). 'Fuzzy sets and topos theory'. *Canadian Math. Bull.*, volume **24**, pages 501–508.

Barr, M. (1990). 'Fixed points in cartesian closed categories'. *Theoretical Computer Science*, volume **70**, pages 65–72.

Barr, M. (1991). '$*$-autonomous categories and linear logic'. *Mathematical Structures in Computer Science*, volume **1**, pages 159–178.

Barr, M. (1995). 'Non-symmetric *-autonomous categories'. *Theoretical Computer Science*, volume **139**, pages 115–130.

Barr, M. (1996). '$*$-autonomous categories, revisited'. *Journal of Pure and Applied Algebra*, volume **111**, pages 1–20.

Barr, M. (1996). 'The Chu construction'. *Theory and Applications of Categories*, volume **2**, pages 17–35.

Barr, M. (1996). 'Separability of tensor in Chu categories of vector spaces'. *Mathematical Structures in Computer Science*, volume **6**, pages 213–217.

Barr, M., C. McLarty, and C. Wells (1985). 'Variable set theory'. Technical report, McGill University.

Barr, M. and C. Wells (1985). *Toposes, Triples and Theories*, volume 278 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, New York. A list of corrections and additions is maintained in [Barr and Wells, 1993].

Barr, M. and C. Wells (1993). 'Corrections to Toposes, Triples and Theories'. Available by anonymous FTP from `triples.math.mcgill.ca` in directory `pub/barr` and by web browser from `http://www.cwru.edu/artsci/math/wells/pub/wellspub.html`. Corrections and additions to [Barr and Wells, 1985].

Barr, M. and C. Wells (1999). *Category Theory for Computing Science, third edition*. Les publications Centre de recherches mathématiques.

Bartha, M. (1992). 'An algebraic model of synchronous systems'. *Information and Computation*, volume **97**, pages 1–31.

Bell, J. L. (1988). *Toposes and Local Set Theories: An Introduction*. Oxford Logic Guides. Oxford University Press.

Bergman, C. and J. Berman (1998). 'Algorithms for categorical equivalence'. *Mathematical Structures in Computer Science*, volume **8**, pages 1–16.

Bird, R. S. (1986). 'An introduction to the theory of lists'. In *Logic of Programming and Calculi of Discrete Designs*, M. Broy, editor, pages 5–42. Springer-Verlag.

Boileau, A. and A. Joyal (1981). 'La logique des topos'. *Symbolic Logic*, volume **46**, pages 6–16.

Borceux, F. (1994). *Handbook of Categorical Algebra I, II and III*. Cambridge University Press.

Chen, H. G. and J. R. B. Cockett (1989). 'Categorical combinators'. Technical report, University of Calgary.

Chu, P.-H. (1978). 'Constructing *-autonomous categories'. Master's thesis, McGill University.

Chu, P.-H. (1979). 'Constructing *-autonomous categories'. Appendix to [Barr, 1979].

Cockett, J. R. B. (1989). 'On the decidability of objects in a locos'. In *Categories in Computer Science and Logic*, J. W. Gray and A. Scedrov, editors, volume 92 of *Contemporary Mathematics*. American Mathematical Society.

Cockett, J. R. B. (1993). 'Introduction to distributive categories'. *Mathematical Structures in Computer Science*, volume **3**, pages 277–307.

Cockett, J. R. B. and D. Spencer (1992). 'Strong categorical datatypes i'. In *Category Theory 1991*, R. Seely, editor. American Mathematical Society.

Corradini, A. and A. Asperti (1993). 'A categorical model for logic programs: Indexed monoidal categories'. In *Semantics: Foundations and Applications (Beekbergen, 1992)*, volume 666 of *Lecture Notes in Computer Science*, pages 110–137. Springer-Verlag.

Cousineau, G., P.-L. Curien, and M. Mauny (1985). 'The categorical abstract machine'. In *Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*. Springer-Verlag.

Crole, R. L. (1994). *Categories for Types*. Cambridge University Press.

Curien, P.-L. (1986). *Categorical Combinators, Sequential Algorithms and Functional Programming*. Wiley.

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

Diers, Y. (1980). 'Catégories localement multiprésentables'. *Arch. Math.*, volume **34**, pages 344–356.

Diers, Y. (1980). 'Quelques constructions de catégories localement multiprésentables'. *Ann. Sci. Math. Québec*, volume **4**, pages 79–101.

Dybjer, P. (1986). 'Category theory and programming language semantics: an overview'. In *Category Theory and Computer Programming*, D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, volume 240 of *Lecture Notes in Computer Science*, pages 165–181. Springer-Verlag.

Dybkjaer, H. and A. Melton (1993). 'Comparing Hagino's categorical programming language and typed lambda-calculi'. *Theoretical Computer Science*, volume **111**, pages 145–189.

Ehrig, H., H. Herrlich, H. J. Kreowski, and G. Preuss, editors (1988). *Categorical Methods in Computer Science*, volume 393 of *Lecture Notes in Computer Science*. Springer-Verlag.

Eilenberg, S. (1976). *Automata, Languages and Machines, Vol. B*. Academic Press.

Eilenberg, S. and J. C. Moore (1965). 'Adjoint functors and triples'. *Illinois J. Math.*, volume **9**, pages 381–398.

Fokkinga, M. M. (1992). 'Calculate categorically!'. *Formal Aspects of Computing*, volume **4**.

Fourman, M. (1977). 'The logic of topoi'. In *Handbook of Mathematical Logic*, J. Barwise, editor. North-Holland.

Fourman, M., P. Johnstone, and A. Pitts (1992). *Applications of Categories in Computer Science*, volume 177 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press.

Fourman, M. and S. Vickers (1986). 'Theories as categories'. In *Category Theory and Computer Programming*, D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, volume 240 of *Lecture Notes in Computer Science*, pages 434–448. Springer-Verlag.

Freyd, P. and A. Scedrov (1990). *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library*. North-Holland.

Girard, J.-Y., P. Taylor, and Y. Lafont (1989). *Proofs and Types*. Cambridge University Press.

Goguen, J. A. (1988). 'What is unification? A categorical view of substitution, equation and solution'. Technical Report CSLI–88–124, Center for the Study of Language and Information.

Goguen, J. A. (1991). 'A categorical manifesto'. *Mathematical Structures in Computer Science*, volume **1**, pages 49–68.

Gray, J. W. and A. Scedrov (1989). *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*. American Mathematical Society.

Gunter, C. (1992). *Semantics of Programming Languages*. The MIT Press.

Hagino, T. (1987). *A categorical programming language*. PhD thesis, University of Edinburgh.

Hagino, T. (1987). 'A typed lambda calculus with categorical type constructors'. In *Category Theory and Computer Science*, D. Pitt, A. Poigné, and D. Rydeheard, editors, volume 283 of *Lecture Notes in Computer Science*, pages 140–157. Springer-Verlag.

Halmos, P. (1960). *Naive Set Theory*. Van Nostrand.

Hindley, J. R. and J. P. Seldin (1986). *Introduction to Combinators and λ-Calculus*. Cambridge University Press.

Huet, G. (1986). 'Cartesian closed categories and lambda-calculus'. In *Combinators and Functional Programming Languages*, G. Cousineau, P.-L. Curien, and B. Robinet, editors, volume 242 of *Lecture Notes in Computer Science*. Springer-Verlag.

Huwig, H. and A. Poigné (1990). 'A note on inconsistencies caused by fixpoints in a cartesian closed category'. *Theoretical Computer Science*, volume **73**, pages 101–112.

Hyland, J. M. E. (1982). 'The effective topos'. In *The L. E. J. Brouwer Centenary Symposium*, pages 165–216. North-Holland.

Hyland, J. M. E. and A. Pitts (1989). 'The theory of constructions: Categorical semantics and topos-theoretic models'. In *Categories in Computer Science and Logic*, J. W. Gray and A. Scedrov, editors, volume 92 of *Contemporary Mathematics*, pages 137–199. American Mathematical Society.

Jacobson, N. (1974). *Basic Algebra I*. W. H. Freeman.

Johnstone, P. T. (1977). *Topos Theory*. Academic Press.

Kleisli, H. (1965). 'Every standard construction is induced by a pair of adjoint functors'. *Proc. Amer. Math. Soc.*, volume **16**, pages 544–546.

Kock, A. (1972). 'Strong functors and monoidal monads'. *Archiv der Mathematik*, volume **20**, pages 113–120.

Lallement, G. (1979). *Semigroups and Combinatorial Applications*. Wiley.

Lambek, J. (1986). 'Cartesian closed categories and typed lambda-calculi'. In *Combinators and Functional Programming Languages*, G. Cousineau, P.-L. Curien, and B. Robinet, editors, volume 242 of *Lecture Notes in Computer Science*, pages 136–175. Springer-Verlag.

Lambek, J. and P. Scott (1984). 'Aspects of higher order categorical logic'. In *Mathematical Applications of Category Theory*, J. W. Gray, editor, volume 30 of *Contemporary Mathematics*, pages 145–174. American Mathematical Society.

Lambek, J. and P. Scott (1986). *Introduction to Higher Order Categorical Logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press.

Lawvere, F. W. (1963). *Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University.

Lawvere, F. W. (1966). 'The category of categories as a foundation for mathematics'. In *Proceedings of the Conference on Categorical Algebra, La Jolla 1965*. Springer-Verlag.

Lewis, H. R. and C. H. Papadimitriou (1981). *Elements of the Theory of Computation*. Prentice-Hall.

Linton, F. E. J. (1969). 'Applied functorial semantics'. In *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Mathematics*, pages 53–74. Springer-Verlag.

Linton, F. E. J. (1969). 'An outline of functorial semantics'. In *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Mathematics*, pages 7–52. Springer-Verlag.

Lüth, C. and N. Ghani (1997). 'Monads and modular term rewriting'. In Moggi and Rosolini [1997], pages 69–86.

Mac Lane, S. (1971). *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag.

Mac Lane, S. and I. Moerdijk (1992). *Sheaves in Geometry and Logic*. Universitext. Springer-Verlag.

Main, M., A. Melton, M. Mislove, and D. Schmidt, editors (1988). *Mathematical Foundations of Programming Language Semantics*, volume 298 of *Lecture Notes in Computer Science*. Springer-Verlag.

Makkai, M. and R. Paré (1990). *Accessible Categories: the Foundations of Categorical Model Theory*, volume 104 of *Contemporary Mathematics*. American Mathematical Society.

Makkai, M. and G. Reyes (1977). *First Order Categorical Logic*, volume 611 of *Lecture Notes in Mathematics*. Springer-Verlag.

Manes, E. G. (1986). 'Weakest preconditions: Categorial insights'. In *Category Theory and Computer Programming*, D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, volume 240 of *Lecture Notes in Computer Science*, pages 182–197. Springer-Verlag.

Manes, E. G. and M. Arbib (1986). *Algebraic Approaches to Program Semantics*. Springer-Verlag.

Marti Oliet, N. and J. Meseguer (1991). 'From Petri nets to linear logic through categories: a survey'. *International Journal of Foundations of Computer Science*, volume **2**, pages 297–399.

McLarty, C. (1989). 'Notes toward a new philosophy of logic'. Technical report, Case Western Reserve University.

McLarty, C. (1992). *Elementary Categories, Elementary Toposes*, volume 21 of *Oxford Logic Guides*. Clarendon Press.

Mikkelson, C. J. (1976). *Lattice Theoretic and Logical Aspects of Elementary Topoi*, volume 25 of *Aarhus University Various Publications Series*. Aarhus University.

Mitchell, J. C. and P. J. Scott (1989). 'Typed lambda calculus and cartesian closed categories'. In *Categories in Computer Science and Logic*, J. W. Gray and A. Scedrov, editors, volume 92 of *Contemporary Mathematics*, pages 301–316. American Mathematical Society.

Moggi, E. (1989). 'Computational lambda-calculus and monads'. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 14–23. IEEE.

Moggi, E. (1991). 'A category-theoretic account of program modules'. *Mathematical Structures in Computer Science*, volume **1**, pages 103–139.

Moggi, E. (1991). 'Notions of computation and monads'. *Information and Control*, volume **93**, pages 155–92.

Moggi, E. and G. Rosolini, editors (1997). *Category Theory and Computer Science, 7th International Conference*. Springer-Verlag.

Mulry, P. S. (1992). 'Strong monads, algebras and fixed points'. In *Applications of Categories in Computer Science*, volume 177 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press.

Pavlović, D. and S. Abramsky (1997). 'Specifying interaction categories'. In Moggi and Rosolini [1997], pages 147–158.

Pierce, B. (1991). *Basic Category Theory for Computer Scientists*. The MIT Press.

Pitt, D. (1986). 'Categories'. In *Category Theory and Computer Programming*, D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, volume 240 of *Lecture Notes in Computer Science*, pages 6–15. Springer-Verlag.

Pitt, D., S. Abramsky, A. Poigné, and D. Rydeheard, editors (1986). *Category Theory and Computer Programming*, volume 240 of *Lecture Notes in Computer Science*. Springer-Verlag.

Pitt, D., P.-L. Curien, S. Abramsky, A. M. Pitts, A. Poigné, and D. E. Rydeheard, editors (1991). *Category Theory and Computer Science (Paris, 1991)*, volume 530 of *Lecture Notes in Computer Science*. Springer-Verlag.

Pitt, D., A. Poigné, and D. Rydeheard, editors (1987). *Category Theory and Computer Science*, volume 283 of *Lecture Notes in Computer Science*. Springer-Verlag.

Pitt, D., D. Rydeheard, P. Dybjer, A. Pitts, and A. Poigné, editors (1989). *Category Theory and Computer Science*, volume 389 of *Lecture Notes in Computer Science*. Springer-Verlag.

Pitt, D., D. E. Rydeheard, and P. Johnstone, editors (1995). *Category Theory and Computer Science, 6th International Conference*. Springer-Verlag.

Poigné, A. (1986). 'Elements of categorical reasoning: Products and coproducts and some other (co-)limits'. In *Category Theory and Computer Programming*, D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, volume 240 of *Lecture Notes in Computer Science*, pages 16–42. Springer-Verlag.

Power, A. J. (1990). 'An algebraic formulation for data refinement'. In *Mathematical Foundations of Programming Semantics*, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, volume 442 of *Lecture Notes in Computer Science*, pages 402–417. Springer-Verlag.

Reynolds, J. C. (1980). 'Using category theory to design implicit conversions and generic operators'. In *Proceedings of the Aarhus Workshop on Semantics-Directed Compiler Generation*, N. D. Jones, editor, volume 94 of *Lecture Notes in Computer Science*. Springer-Verlag.

Rydeheard, D. E. and R. M. Burstall (1985). 'Monads and theories: A survey for computation'. In *Algebraic Methods in Semantics*, M. Nivat and J. C. Reynolds, editors. Cambridge University Press.

Rydeheard, D. E. and R. M. Burstall (1986). 'A categorical unification algorithm'. In *Category Theory and Computer Programming*, D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, volume 240 of *Lecture Notes in Computer Science*, pages 493–505. Springer-Verlag.

Rydeheard, D. E. and R. M. Burstall (1988). *Computational Category Theory*. International Series in Computer Science. Prentice Hall.

Scott, D. (1972). 'Continuous lattices'. In *Toposes, Algebraic Geometry and Logic*, F. W. Lawvere, editor, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer-Verlag.

Scott, D. (1982). 'Domains for denotational semantics'. In *Automata, Languages and Programming*, M. Nielson and E. M. Schmidt, editors, volume 140 of *Lecture Notes in Computer Science*, pages 577–613. Springer-Verlag.

Sedgewick, R. (1983). *Algorithms*. Addison-Wesley.

Seely, R., editor (1992). *Category Theory 1991*, volume 13 of *Canadian Mathematical Society Conference Proceedings*. American Mathematical Society.

Smyth, M. B. (1983). 'The largest cartesian closed category of domains'. *Theoretical Computer Science*, volume **27**.

Spivey, M. (1989). 'A categorical approach to the theory of lists'. In *Mathematics of Program Construction*, J. L. A. Van de Snepscheut, editor, volume 375 of *Lecture Notes in Computer Science*, pages 399–408. Springer-Verlag.

Taylor, P. (1989). 'Quantitative domains, groupoids and linear logic'. In *Category Theory and Computer Science*, D. Pitt, D. Rydeheard, P. Dybjer, A. Pitts, and A. Poigné, editors, volume 389 of *Lecture Notes in Computer Science*. Springer-Verlag.

Tennent, R. D. (1986). 'Functor category semantics of programming languages and logics'. In *Category Theory and Computer Programming (Guildford, 1985)*, D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, volume 240 of *Lecture Notes in Computer Science*, pages 206–224. Springer-Verlag.

Vickers, S. (1992). 'Geometric theories and databases'. In *Applications of Categories in Computer Science (Durham, 1991)*, volume 177 of *London Mathematical Society Lecture Notes Series*, pages 288–314. Cambridge University Press.

Wadler, P. (1989). 'Theorems for free!'. In *Fourth ACM Symposium on Functional Programming Languages and Computer Architecture*, pages 347–359. Association for Computing Machinery.

Wadler, P. (1992). 'Comprehending monads'. *Mathematical Structures in Computer Science*, volume **2**, pages 461–493.

Wagner, E. G. (1986). 'Algebraic theories, data types and control constructs'. *Fundamenta Informatica*, volume **9**, pages 343–370.

Wagner, E. G. (1986). 'Categories, data types and imperative languages'. In *Category Theory and Computer Programming*, D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, volume 240 of *Lecture Notes in Computer Science*, pages 143–162. Springer-Verlag.

Wagner, E. G. (1987). 'A categorical treatment of pre- and post conditions'. *Theoretical Computer Science*, volume **53**.

Walters, R. F. C. (1991). *Categories and Computer Science*. Cambridge University Press.

Walters, R. F. C. (1992). 'An imperative language based on distributive categories'. *Mathematical Structures in Computer Science*, volume **2**, pages 249–256.

Williams, J. H. (1982). 'Notes on the fp style of functional programming'. In *Functional Programming and its Applications*, P. H. Darlington, J. and D. Turner, editors. Cambridge University Press.

Index