A Gentle Introduction to Category Theory

— the calculational approach —

Maarten M. Fokkinga

Version of June 6, 1994

Maarten M. Fokkinga
University of Twente, dept. INF
PO Box 217
NL 7500 AE ENSCHEDE
The Netherlands

e-mail: `fokkinga@cs.utwente.nl`

Contents

Chapters 3 and 5 of 'Law and Order in Algorithmics' [4] present a categorical approach to algebras. Those chapters don't use the notions of adjunction and colimit. So you may skip Sections 1d, and 2e, and Appendix A when you are primarily interested in reading those chapters.

Chapter 0

Introduction

0.1 Aim. In these notes we present the important notions from category theory. The intention is to provide a fairly good skill in manipulating with those concepts formally. What you probably will not acquire from these notes is the ability to recognise the concepts in your daily work when that differs from algorithmics, since we give only a few examples and those are taken from algorithmics. For such an ability you need to work through many, very many examples, in diverse fields of applications.

This text differs from most other introductions to category theory in the calculational style of the proofs (especially in Chapter 2 and Appendix A), the restriction to applications within algorithmics, and the omission of many additional concepts and facts that I consider not helpful in a first introduction to category theory.

0.2 Acknowledgements. This text is a compilation and extension of work that I've done for my thesis. That project would have been a failure without the help or stimulation by many people. Regarding the technical contents, Roland Backhouse, Grant Malcolm, Lambert Meertens and Jaap van der Woude may recognise their ideas and methodological and notational suggestions.

0.3 Why category theory? There are various views on what category theory is about, and what it is good for. Here are some.

- Category theory is a relatively young branch of mathematics, stemming from algebraic topology, and designed to describe various *structural* concepts from different mathematical fields in a *uniform* way. Indeed, category theory provides a bag of concepts (and theorems about those concepts) that form an abstraction of many concrete concepts in diverse branches of mathematics, including computing science.

  Hence it will come as no surprise that the concepts of category theory form an abstraction of many concepts that play a role in algorithmics.

- Quoting Hoare [6]: "Category theory is quite the most general and abstract branch of pure mathematics. [...] The corollary of a high degree of generality and abstraction is that the theory gives almost no assistance in solving the more specific problems within any of the subdisciplines to which it applies. It is a tool for the generalist, of little benefit to the practitioner [. . .]."

Hence it will come as no surprise that, for algorithmics too, category is mainly useful for theory development; hardly for individual program derivation.

- Quoting Asperti and Longo [1]: "Category theory is a mathematical jargon. [. . .] Many different formalisms and structures may be proposed for what is essentially the same concept; the categorical language and approach may simplify through abstraction, display the generality of concepts, and help to formulate uniform definitions."

- Quoting Scott [7]: "[Category theory offers] a pure theory of functions, not a theory of functions derived from sets."

  To this I want to add that the language of category theory facilitates an elegant style of expression and proof (equational reasoning); for the use in algorithmics this happens to be reasoning at the function level, without the need (and the possibility) to introduce arguments explicitly. Also, the formulas often suggest and ease a far-reaching generalisation, much more so than the usual set-theoretic formulations.

Category theory has itself grown to a branch in mathematics, like algebra and analysis, that is studied like any other one. One should not confuse the potential benefits that category theory may have (for the theory underlying algorithmics, say) with the difficulty and complexity, and fun, of doing category theory as a specialisation in itself.

0.4 Preliminaries: sequences. Our examples frequently involve finite lists, or sequences as we like to call them. Here is our notation.

A **sequence** is a finite list of elements of a certain type, denoted $[a_0, \ldots, a_{n-1}]$. The set of sequences over A is denoted $Seq\, A$. Further operations are:

$$
\begin{array}{lcl}
tip & = & a \mapsto [a] \\
 & : & A \to Seq\, A \\
\_ : \_ & = & (a, [a_0, \ldots, a_{n-1}]) \mapsto [a, a_0, \ldots, a_{n-1}] \\
 & : & A \times Seq\, A \to Seq\, A \\
cons & = & \text{prefix written operation } \_ : \_ \\
\_ +\!\!+ \_ & = & ([a_0, \ldots, a_{m-1}], [a_m, \ldots, a_{n-1}]) \mapsto [a_0, \ldots, a_{n-1}] \\
 & : & Seq\, A \times Seq\, A \to Seq\, A \\
join & = & \text{prefix written operation } \_ +\!\!+ \_ \\
Seq\, f & = & [a_0, \ldots, a_{n-1}] \mapsto [f\, a_0, \ldots, f\, a_{n-1}] \\
 & : & Seq\, A \to Seq\, B \ \text{ whenever } \ f\colon A \to B \\
\oplus/ & = & [a_0, \ldots, a_{n-1}] \mapsto a_0 \oplus \ldots \oplus a_{n-1} \\
 & : & Seq\, A \to A \ \text{ whenever } \ \oplus\colon A \times A \to A \ \text{ and}
\end{array}
$$

$$\oplus \text{ is associative and has a neutral element}$$

Function $Seq\, f$ is often called **map** f. Function $\oplus/$ is called the **reduce-with-**$\oplus$ or the **fold-with-**$\oplus$; the neutral element of $\oplus$ is the outcome on the empty sequence $[\,]$. Associativity of $\oplus$ implies that the specification of $\oplus/$ is unambiguous, not depending on the parenthesisation within $a_0 \oplus \ldots \oplus a_{n-1}$.
Exercise: find a non-associative operation for which $\oplus/$ is well defined. Conclude that associativity of $\oplus$ is a sufficient, but not necessary condition for $\oplus/$ to be a well defined function on sequences.

You may familiarise yourself with these operations by proving the laws listed in paragraph 1.49 on account of the above definitions (noting that $f\, ; g = g \circ f =$ the composition 'f followed by g').

Chapter 1

The main concepts

This introductory chapter gives a brief overview of the important categorical concepts, namely *category, functor, naturality, adjunction, duality*. In the next chapter we will show how to express familiar set-theoretic notions in category theoretic terms.

1a Categories

A category is a collection of data that satisfy some particular properties. So, saying that *such-and-so* forms a category is merely short for asserting that *such-and-so* satisfy all the axioms of a category. Since a large body of concepts and theorems have been developed that are based on the categorical axioms only, those concepts and theorems are immediately available for *such-and-so* if that forms a category.

For an intuitive understanding in the following definition, one may interpret objects as sets, and morphisms as typed total functions. We shall later provide some more and quite different examples of a category, in which the objects aren't sets and the morphisms aren't functions.

1.1 Definition. A **category** is: the following data, subject to the axioms listed in paragraph 1.2.

- A collection of things called **objects**.
  By default, $A, B, C, \ldots$ vary over objects.
- A collection of things called **morphisms**, sometimes called **arrows**.
  By default, $f, g, h, \ldots$, and later on also $\alpha, \beta, \varphi, \psi, \chi, \ldots$, vary over morphisms.
- A relation on morphisms and pairs of objects, called **typing** of the morphisms.
  By default, the relation is denoted $f\colon A \to B$, for morphism f and objects A, B. In this case we also say that $A \to B$ is the **type** of f, and that f is a morphism **from** A **to** B. In view of the axioms below we may define the **source** and **target**

by

$$\operatorname{src} f = A \text{ and } \operatorname{tgt} f = B \qquad \text{whenever } f\colon\ A \to B\,.$$

- A binary partial operation on morphisms, called **composition**.
  By default, $f \mathbin{;} g$ is the notation of the composition of morphisms f and g. An alternative notation is $g \circ f$, and even $g\,f$, with the convention $f \mathbin{;} g = g \circ f = g\,f$. Within a term denoting a morphism, symbol $;$ has **weakest binding** power, whereas juxtaposition binds strongest. We shall hardly use symbol $\circ$ to denote composition.

- For each object A a distinguished morphism, called **identity** on A.
  By default, id_A, or id when A is clear from the context, denotes the identity on object A.

By default, $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ vary over categories, and particular categories are named after their objects (rather than their morphisms). Actually, these data define the basic terms of the **categorical language** in which properties of the category can be stated. A categorical statement is an expression built from (notations for) objects, typing, morphisms, composition and identities by means of the usual logical connectives and quantifications and equality. If you happen to know what the objects really are, you may use those aspects in your statements, but then you are *not* expressing yourself categorically.

Sometimes there are several categories under discussion. Then the name of the category may and must be added to the above notations, as a subscript or otherwise, in order to avoid ambiguity. So, let $\mathcal{A}$ be a category. Then we may write specifically $f\colon\ A \to_{\mathcal{A}} B$, $\operatorname{src}_{\mathcal{A}}$, $\operatorname{tgt}_{\mathcal{A}}$, $f \mathbin{;_{\mathcal{A}}} g$, and $id_{\mathcal{A},A}$. There is no requirement in the definition of a category stating that the morphisms of one should be different from those of another; a morphism of $\mathcal{A}$ may also be a morphism of $\mathcal{B}$. In such a case the indication of $\mathcal{A}$ in $f\colon\ A \to_{\mathcal{A}} B$ and $\operatorname{src}_{\mathcal{A}} f$ is quite important.

1.2 Axioms. There are three 'typing' axioms, and two axioms for equality. The typing axioms are these:

1.3 $\quad f\colon\ A \to B$ and $f\colon\ A' \to B' \quad \Rightarrow \quad A = A'$ and $B = B'$ $\qquad$ unique-Type

1.4 $\quad f\colon\ A \to B$ and $g\colon\ B \to C \quad \Rightarrow \quad f \mathbin{;} g :\ \ A \to C$ $\qquad$ composition-Type

1.5 $\quad id_A\colon\ A \to A$ $\qquad$ identity-Type

A morphism term f is **well-typed** if: a typing $f\colon\ A \to B$ can be derived for some objects A, B according to these axioms (and the Type properties of defined notions that will be given in the sequel).

Convention. Whenever we write a term, we assume that the variables are typed (at their introduction — mostly an implicit universal quantification in front of the formula) in such a way that the term is well-typed. This convention allows us to simplify the formulations considerably, as illustrated in the following axioms.

Here are the two axioms for equality of morphisms.

1.6 $(f \mathbin{;} g) \mathbin{;} h = f \mathbin{;} (g \mathbin{;} h)$ composition-Assoc

1.7 $id \mathbin{;} f = f = f \mathbin{;} id$ Identity

In accordance with the convention explained a few lines up, axiom composition-Assoc is universally quantified with "for all objects A, B, C, D and all morphisms f, g, h with $f\colon A \to B$, $g\colon B \to C$, and $h\colon C \to D$", or slightly simpler, "for all f, g, h with $\mathrm{tgt}\, f = \mathrm{src}\, g$ and $\mathrm{tgt}\, g = \mathrm{src}\, h$". In accordance with that same convention, axiom Identity actually reads $id_{\mathrm{src} f} \mathbin{;} f = f = f \mathbin{;} id_{\mathrm{tgt} f}$, or even "for all objects A, B and all morphisms f with $f\colon A \to B$, $id_A \mathbin{;} f = f = f \mathbin{;} id_B$".

Convention. The category axioms are so basic that we shall mostly use them tacitly. In particular, we shall use composition-Assoc implicitly by omitting the parentheses in a composition, thus writing $f \mathbin{;} g \mathbin{;} h$ instead of either $(f \mathbin{;} g) \mathbin{;} h$ or $f \mathbin{;} (g \mathbin{;} h)$.

1.8 Pre-category. If the requirement unique-Type is dropped in the definition of a category, then one gets the definition of a **pre-category**.

Quite often we shall encounter data that form a pre-category. By a simple trick, those data also determine a category: take multiple copies of the morphisms and make them distinct by incorporating a "source" and "target" into them. Formally, let $\mathcal{A}$ be a pre-category, and define $\mathcal{B}$ by

an object in $\mathcal{B}$	is:	an object in $\mathcal{A}$
a morphism in $\mathcal{B}$	is:	a triple (A, f, B) with $f\colon A \to_{\mathcal{A}} B$
$f\colon A \to_{\mathcal{B}} B$	$\equiv$	$A = A'$ and $B = B'$ where $(A', f', B') = f$
$f \mathbin{;_{\mathcal{B}}} g$	$=$	$(A,\ f' \mathbin{;_{\mathcal{A}}} g',\ C)$
		where $(A, f', B) = f$ and $(B, g', C) = g$
$id_{\mathcal{B},A}$	$=$	$(A, id_{\mathcal{A},A}, A)$.

Then $\mathcal{B}$ is a category. (Exercise: prove this.)

In the sequel, we shall sometimes pretend that a pre-category *is* a category, that is, we shall define a category out of it by the above construction, but keep writing f instead of (A, f, B) for the morphisms.

A big technical advantage of categories over pre-categories is that there is no need to specify the source and target of a morphism; they are determined by morphism f as $\mathrm{src}\, f$ and $\mathrm{tgt}\, f$, respectively. (Nevertheless we shall often explicitly name the source and target of a morphism, for clarity.) A big conceptual advantage of pre-categories over categories is that the morphisms more closely correspond to the structure preserving transformations of interest. It seems that most concepts and theorems for categories can be generalised to pre-categories.

1.9 Example: $\mathcal{S}et$. $\mathcal{S}et'$ is: the pre-category whose objects are sets, whose morphisms are total functions, and whose composition and identities are function composition and identity functions respectively. Further, define $f\colon A \to B$ to mean that, for each $a \in A$, fa is well-defined and $fa \in B$. Thus, for the squaring function *square* we have $square\colon nat \to nat$ as well as $square\colon real \to real$, and so on. With this definition the axioms listed in paragraph 1.2, except for unique-Type, are fulfilled. (Exercise: verify the axioms.)

Now define category $\mathcal{S}et$ out of pre-category $\mathcal{S}et'$ by the construction given in paragraph 1.8. We keep saying that the morphisms in $\mathcal{S}et$ are total functions; it may be more accurate to say that they are 'typed' total functions, since they carry their type (source and target) with them. We also keep the notation fa for the application of f on a, whenever $f\colon A \to_{\mathcal{S}et} B$ and $a \in A$.

Doing set theory in the categorical language enforces the strait jacket of expressing everything with function composition only, without using explicit arguments, membership and function application. Once mastered it is often, not always, an elegant way of expressing.

We shall mostly illustrate the notions of category theory in terms of categories where the morphisms are functions and composition is function composition, like in $\mathcal{S}et$. But beware, even if the morphisms are functions, the objects need not at all be sets: sometimes the objects are *operations* (with an appropriate definition for the typing of the functions). The categories of F-algebras are an example of this; a special case is $\mathcal{A}lg(\mathbb{I})$ discussed in paragraph 1.22, and $\mathcal{M}on$ in paragraph 1.23. Other times we'll take "structures" (of structured data) as objects, again with an appropriate interpretation for the typing of the morphisms; this occurs in category $\mathcal{F}tr(\mathcal{A}, \mathcal{B})$ defined in paragraph 1.37.

1.10 Example: graphs and pre-orders. One should not be mislead by our illustrations, where morphisms *are* functions. There are many more mathematical data that can be viewed as a category. To mention just one generic example, each directed graph determines a category as follows. Take the nodes as objects, and all paths as morphisms typed with their start and end nodes. Composition is concatenation of paths, and the identities are the empty paths. Thus defined, these data do satisfy the axioms listed in paragraph 1.2, hence form a category. (Exercise: verify this.)

Here is yet another important example of a class of categories. We don't need it in our discussion of algorithmics, but it provides sometimes instructive examples. Each pre-ordered set $(A, \leq)$ can be considered a category, in the following way. The elements $a, b, \ldots$ of A are the objects of the category and there is a morphism from a to b precisely when $a \leq b$. Formally, the category is defined as follows.

$$\begin{array}{lll}
\text{an object} & \text{is:} & \text{an element in } A \\
\text{a morphism} & \text{is:} & \text{a pair } (a,b) \text{ with } a \leq b \text{ in } A \\
(a,b)\colon c \to d & \equiv & a = c \quad \wedge \quad b = d \\
(a,b) \,;\, (b,c) & = & (a,c) \\
id_a & = & (a,a)\,.
\end{array}$$

Thus defined, these data do satisfy the axioms of a category.
Exercise: check that laws composition-Assoc and -Identity are satisfied, and that the typing axioms follow from the definition of the typing relation, and that the well-definedness of $;$ and id follow from the transitivity and reflexivity of the preorder $\leq$, respectively.
Exercise: define a category where the morphisms are numbers, and the composition is addition.

1.11 Cartesian closed categories, and Topoi. The axioms on the morphisms and composition are very weak, so that many mathematical structures can be rendered as a category. By imposing extra axioms, still in the categorical language, the categories may have more of the properties you are actually interested in.

For example, a *cartesian closed category* is a category in which the extra properties make the morphisms behave more like real *functions*: in particular, there is a notion of *currying* and of *applying* a curried morphism. There is a close relationship between this type of categories and *typed* λ*-calculi*.

As another example, a *topos* (plural: topoi or toposes) is a cartesian closed category in which the extra properties make the objects have more of the properties of real sets: in particular, for each object there exists an object of its 'subobjects'.

In these notes, we shall nowhere need the extra properties. As a result, the notions defined here, and the theorems proved, are very general and very weak at the same time.

1.12 Discussion. Quoting Asperti and Longo [1]: "The basic premise of category theory is that every kind of mathematically structured object comes equipped with a notion of [. . .] transformation, called 'morphism', that preserves the structure of the object." By means of the categorical language one cannot express properties of the *internal* structure of objects. The internal structure of objects is accessible only *externally* through the morphisms between the objects. The morphisms may seem (and sometimes are) functions, but the categorical language doesn't express that; it only provides a way to express the composition of morphisms. The discipline of expressing internal structure only externally is the key to the *uniformity* of describing structural concepts in various different application fields.

Since each graph is a category, the above interpretation of "internal structure" of objects doesn't always make sense.

Exercise. What functions are precisely the functions that preserve a partial order? (Define a category in which these functions are the morphisms.) What functions are precisely the functions that preserve a partial order and the limits? (Define a category in which these functions are the morphisms.) What structure of sets is preserved by precisely all total functions? (What category has these functions as morphisms?)

1.13 Expressing concepts categorically. In the following chapters we shall show how all kinds of familiar concepts can be expressed categorically, that is, using only the notions of object, morphism, typing, composition, identity, and notions that can be defined in terms of these, and further the usual logical connectives and quantifications.

To appreciate the problems and delight involved, the reader may spend some (not too much) time in trying to find a categorically expressed property P such that in $\mathcal{S}et$ property P holds precisely for the empty set, that is, $P(A) \equiv A{=}\emptyset$. Similarly, you may think about categorically expressed properties P such that in $\mathcal{S}et$ the following holds:

$$
\begin{array}{lcl}
P(A) & \equiv & A = \{17\} \\
P(A) & \equiv & A \text{ is a singleton set} \\
P(A,B) & \equiv & A \subseteq B \\
P(f) & \equiv & f \text{ is surjective} \\
P(f) & \equiv & f \text{ is injective} \\
P(A,B,C) & \equiv & C = A \cup B \\
P(A,B,C,f,g) & \equiv & C \text{ is the disjoint union of } A \text{ and } B \\
 & & \text{with injections } f\colon A \to C \text{ and } g\colon B \to C\ .
\end{array}
$$

Also, think about a way to *represent* a binary relation R on A categorically; what collection of sets and functions may carry the same information as R?

Just for fun you may also think about categorically expressed properties P such that in a pre-order considered as a category (see paragraph 1.10) the following holds:

$$
\begin{array}{lcl}
P(a) & \equiv & a \text{ is a least element} \\
P(a) & \equiv & a \text{ is a greatest element} \\
P(a,b,c) & \equiv & c \text{ is a greatest lower bound of } a \text{ and } b\ ,
\end{array}
$$

and what is the interpretation of these properties in $\mathcal{S}et$?

1.14 Constructing new categories. There are several ways in which new categories can be constructed out of given ones. Here, we give just two ways, and in paragraph 1.24 we'll sketch some other ways.

A subcategory of $\mathcal{A}$ is completely determined by its objects and morphisms, and $\mathcal{A}$. Formally, a **subcategory** of a category $\mathcal{A}$ is: a category in which each object, morphism, and identity is an object, morphism, and identity in $\mathcal{A}$, and in which the typing and composition is the typing and composition of $\mathcal{A}$ restricted to the objects and morphisms of the subcategory.

A full subcategory of $\mathcal{A}$ is completely determined by its objects, and $\mathcal{A}$. Formally, a subcategory of a category $\mathcal{A}$ is a **full** subcategory of $\mathcal{A}$ if: for each A, B in the subcategory, *all* the morphisms with type $A \to B$ in $\mathcal{A}$ are morphisms in the subcategory.

A category is **built upon** a category $\mathcal{A}$ if: its morphisms are morphisms in $\mathcal{A}$, and the composition and identities are inherited from $\mathcal{A}$, and further, its objects are *collections of morphisms* of $\mathcal{A}$, and its typing $f\colon A \to B$ is defined as a *collection of equations* between the morphisms f, A, B in $\mathcal{A}$. An example is spelled out in detail in paragraph 1.22: category $\mathcal{A}lg(I\!I)$ is built upon $\mathcal{S}et$, and has binary operations as objects and homomorphisms as morphisms. Also, in paragraph 1.23 category $\mathcal{M}on$ will be defined as a subcategory of $\mathcal{A}lg(I\!I)$, and hence $\mathcal{M}on$ is built upon $\mathcal{S}et$ too.

Exercise: prove that 'being a subcategory of' is a partial order: reflexive, antisymmetric, and transitive. Also, prove that a subcategory of a category built upon $\mathcal{A}$ is itself built upon $\mathcal{A}$.

1b Functors

A functor is a mapping from one category to another that preserves the categorical structure, that is, it preserves the property of being an object, the property of being a morphism, the typing, the composition, and the identities. The significance of functors is manifold: they map one mathematical structure (category, piece of mathematics) to another, they turn up as *objects* of interesting categories, they are *the* mathematically obvious type of transformation between categories, and, last but not least, they form a categorical tool to deal with "structured" objects (as we shall explain in paragraph 1.21).

1.15 Definition. Let $\mathcal{A}$ and $\mathcal{B}$ be categories; then a **functor** from $\mathcal{A}$ to $\mathcal{B}$ is: a mapping F that sends objects of $\mathcal{A}$ to objects of $\mathcal{B}$, and morphisms of $\mathcal{A}$ to morphisms of $\mathcal{B}$ in such a way that

1.16	Ff	:	$FA \to_{\mathcal{B}} FB$	whenever $f\colon A \to_{\mathcal{A}} B$	ftr-Type
1.17	$F id_A$	=	id_{FA}	for each object A in $\mathcal{A}$	Functor
1.18	$F(f\,;g)$	=	$Ff\,;Fg$	whenever $f\,;g$ is well-typed	Functor

Formula $F\colon \mathcal{A} \to \mathcal{B}$ means that F is a functor from $\mathcal{A}$ to $\mathcal{B}$, and we say that $\mathcal{A} \to \mathcal{B}$ is a (the) **type** of F. An **endo**functor is: a functor of type $\mathcal{A} \to \mathcal{A}$, for some $\mathcal{A}$; its source and target are equal. By default, $F, G, H, J, \ldots$ vary over functors.

The two axioms Functor are equivalent to the single statement that functors distribute over finite compositions:

$$F(f\,;\ldots;g) \quad = \quad Ff\,;\ldots;Fg\,,$$

with id being the empty composition.

1.19 Example: functor $I\!I$. Consider category $\mathcal{S}et$. Define mapping $I\!I$ (pronounced *twin*, or *bin* from binary) as follows.

$$\begin{array}{lcll} I\!I\,A & = & A \times A & \text{a set, hence object in } \mathcal{S}et \\ I\!I\,f & = & (a, a') \mapsto (fa, fa') & \\ & : & I\!I\,A \to I\!I\,B & \text{whenever } f\colon A \to B\,. \end{array}$$

For example, $I\!I\,nat$ is the set of pairs of natural numbers, and $I\!I\,succ$ maps $(19,48)$ onto $(20,49)$. Mapping $I\!I$ satisfies the functor properties; it is a functor $I\!I$: $\mathcal{S}et \to \mathcal{S}et$. (Exercise: verify the functor axioms.) Functor $I\!I$ can be used to characterise *bin*ary operations in a neat way. For example,

$$\begin{array}{lcl} + & : & I\!I\,nat \to nat \\ n \mapsto (n \operatorname{div} 10, n \bmod 10) & : & nat \to I\!I\,nat\,. \end{array}$$

We shall say that the binary operations are $I\!I$-ary operations.

1.20 Example: functor Seq**.** Mapping Seq discussed in paragraph 0.4 is a functor with type $\mathcal{S}et \to \mathcal{S}et$. To see this, recall that:

$$\begin{array}{lcll} \mathit{Seq}\,A & = & \text{the set of finite sequences over } A\,, & \text{an object in } \mathcal{S}et \\ \mathit{Seq}\,f & = & [a_0,\ldots,a_{n-1}] \mapsto [fa_0,\ldots,fa_{n-1}] & \\ & : & \mathit{Seq}\,A \to \mathit{Seq}\,B & \text{whenever } f\colon\ A \to B\,. \end{array}$$

Property ftr-Type is the second line of the above equation for $\mathit{Seq}\,f$, and the equations $\mathit{Seq}\,id_A = id_{\mathit{Seq}\,A}$ and $\mathit{Seq}(f\,;\,g) = \mathit{Seq}\,f\,;\,\mathit{Seq}\,g$ are easily verified. (Exercise: do this.) Functions on or to sequences have Seq in their source or target type, respectively. For example, function rev_A that reverses sequences over A, has type $\mathit{Seq}\,A \to \mathit{Seq}\,A$.

1.21 A use of functors. In the definition of a category, objects are "just things" for which no internal structure is observable *by categorical means* (composition, identities, morphisms, and typing). Functors form the tool to deal with "structured" objects. Indeed, in $\mathcal{S}et$ an (the?) aspect of a structure is that it has "constituents", and that it is possible to apply a function to all the individual constituents; this is done by $Ff\colon\ FA \to FB$. So $I\!I$ is or represents the structure of pairs; $I\!I A$ is the set of pairs of A, and $I\!I f$ is the functions that applies f to each constituent of a pair. Also, Seq is or represents the structure of sequences; $\mathit{Seq}\,A$ is the structure of sequences over A, and $\mathit{Seq}\,f$ is the function that applies f to each constituent of a sequence.

Even though FA is still just an object, a thing with no observable internal structure, the functor properties enable to exploit the "structure" of FA. The following example may make this clear; it illustrates how functor $I\!I$ is or represents the *structure of pairs*. It illustrates at the same time where and how the functor properties play a rôle.

For this, let $\oplus\colon\ I\!I A \to A$ and $\otimes\colon\ I\!I B \to B$ be *bin*ary operations on sets A and B respectively, and let $f\colon\ A \to B$ be a function. We define the notation

$$f \quad : \quad \oplus \to_{I\!I} \otimes$$

to mean

$$\oplus\,;\,f \quad = \quad I\!I f\,;\,\otimes\,.$$

Expressed in set-theoretic terms, property $f\colon \oplus \to_{I\!I} \otimes$ means that $f(x \oplus y) = f\,x \otimes f\,y$ for all x, y in the source set of $\oplus$. Following mathematical terminology we call such a function f a **homomorphism** from $\oplus$ to $\otimes$.

Now, property ftr-Type implies well-typedness of the defining equation of $\to_{I\!I}$, independently of the actual meaning of $I\!I$. (Exercise: verify this claim.) The two other properties, Functor, enable us to prove the following theorem independently of the actual meaning of $I\!I$. The theorem expresses that for each operation the identity is a homomorphism from that operation to itself, and that the composition of homomorphisms is a homomorphism again.

Theorem

$$id\colon \oplus \to_{I\!I} \oplus$$

$$f\colon \oplus \to_{I\!I} \otimes \text{ and } g\colon \otimes \to_{I\!I} \odot \quad \Rightarrow \quad f \,;\, g\colon \oplus \to_{I\!I} \odot\,.$$

Indeed, for the former we argue

$$\begin{array}{ll}
 & id\colon \oplus \to_{I\!I} \oplus \\
\equiv & \qquad \text{definition } \to_{I\!I} \\
 & \oplus \,;\, id = I\!I\,id \,;\, \oplus \\
\equiv & \qquad \text{lhs: Identity} \\
 & id \,;\, \oplus = I\!I\,id \,;\, \oplus \\
\Leftarrow & \qquad \text{Leibniz} \\
 & id = I\!I\,id \\
\equiv & \qquad \text{Functor} \\
 & true.
\end{array}$$

For the latter we argue

$$\begin{array}{ll}
 & f \,;\, g\colon \oplus \to_{I\!I} \odot \\
\equiv & \qquad \text{definition } \to_{I\!I} \\
 & \oplus \,;\, f \,;\, g = I\!I(f \,;\, g) \,;\, \odot \\
\equiv & \qquad \text{lhs: premise } f\colon \oplus \to_{I\!I} \otimes\text{, and definition } \to_{I\!I} \\
 & I\!I f \,;\, \otimes \,;\, g = I\!I(f \,;\, g) \,;\, \odot \\
\equiv & \qquad \text{lhs: premise } g\colon \otimes \to_{I\!I} \odot\text{, and definition } \to_{I\!I} \\
 & I\!I f \,;\, I\!I g \,;\, \odot = I\!I(f \,;\, g) \,;\, \odot \\
\Leftarrow & \qquad \text{Leibniz} \\
 & I\!I f \,;\, I\!I g = I\!I(f \,;\, g) \\
\equiv & \qquad \text{Functor} \\
 & true.
\end{array}$$

Not only is the actual meaning of $I\!I$ nowhere used, but also it is nowhere used that A, B are sets (there is nowhere a membership $\in$) or that $f, g, \oplus, \ldots$ are functions or operations, respectively. Only the category axioms (all except unique-Type) and the functor

axioms (all of them) have been used. So the above definition, theorem, and proof are valid for any functor and any category, not just for functor $\mathit{I\!I}$ and category $\mathcal{S}et$. Here we see how a categorical formulation suggests or eases a far-reaching generalisation: replace $\mathit{I\!I}$ by an arbitrary functor F, and you have a definition of 'F-ary operation' and 'F-homomorphism', and a theorem together with its proof about that notion, and these are valid for an arbitrary category.
Exercise: generalise the above theorem and proof by replacing $\mathit{I\!I}$ everywhere by an arbitrary functor F; check each step. Also, generalise from $\mathcal{S}et$ to an arbitrary category.

This concludes an illustration of the use of the functor axioms, and of using functors to deal with "structured objects".

1.22 Category $\mathcal{A}lg(\mathit{I\!I})$. The theorem in paragraph 1.21 gives rise to another category, to be called $\mathcal{A}lg(\mathit{I\!I})$; an important one for algorithmics, as will become clear in the sequel. In words, $\mathcal{A}lg(\mathit{I\!I})$ has the $\mathit{I\!I}$-ary operations in $\mathcal{S}et$ as objects, the homomorphisms for these operations as morphisms, and it inherits the composition and identities from $\mathcal{S}et$. (This fixes everything except the typing.) Formally, $\mathcal{A}lg(\mathit{I\!I})$ is defined thus:

an $\mathcal{A}lg(\mathit{I\!I})$-object	is:	a $\mathit{I\!I}$-ary operation in $\mathcal{S}et$
an $\mathcal{A}lg(\mathit{I\!I})$-morphism	is:	a homomorphism for $\mathit{I\!I}$-ary operations, in $\mathcal{S}et$
$f\colon \oplus \to_{\mathcal{A}lg(\mathit{I\!I})} \otimes$	$\equiv$	$f\colon \oplus \to_{\mathit{I\!I}} \otimes$
	$\equiv$	$\oplus \,;\, f = \mathit{I\!I} f \,;\, \otimes$
$f \,;_{\mathcal{A}lg(\mathit{I\!I})}\, g$	$=$	$f \,;_{\mathcal{S}et}\, g$
$id_{\mathcal{A}lg(\mathit{I\!I}),\oplus}$	$=$	$id_{\mathcal{S}et,A}$ where $A = \mathrm{tgt}_{\mathcal{S}et}(\oplus)$.

Thus, $\mathcal{A}lg(\mathit{I\!I})$ is built upon $\mathcal{S}et$ (see paragraph 1.14 for 'built upon'). Let us see whether the category axioms are fulfilled for $\mathcal{A}lg(\mathit{I\!I})$. The theorem in paragraph 1.21 asserts that axioms composition-Type and identity-Type are fulfilled. The axioms composition-Assoc and Identity are fulfilled since the composition and the identities are inherited from category $\mathcal{S}et$. So, $\mathcal{A}lg(\mathit{I\!I})$ is at least a pre-category. With the definition above axiom unique-Type is not fulfilled so that $\mathcal{A}lg(\mathit{I\!I})$ is not a category. The reason is that a function can be a homomorphism for several distinct operations, that is, $f\colon \oplus \to_{\mathit{I\!I}} \otimes$ and $f\colon \oplus' \to_{\mathit{I\!I}} \otimes'$ can both be true while the pair $\oplus, \otimes$ differs from the pair $\oplus', \otimes'$. (Exercise: find such a function and operations.)

In the sequel we shall pretend that $\mathcal{A}lg(\mathit{I\!I})$ is made into a category (re-defining it) by the technique of constructing a category out of a pre-category, see paragraph 1.8. Thus, $f\colon \oplus \to_{\mathcal{A}lg(\mathit{I\!I})} \otimes$ denotes the typing in $\mathcal{A}lg(\mathit{I\!I})$, and implies that $\mathrm{tgt}_{\mathcal{A}lg(\mathit{I\!I})} f = \oplus$, whereas formula $f\colon \oplus \to_{\mathit{I\!I}} \otimes$ keeps to mean only that $\oplus \,;\, f = \mathit{I\!I} f \,;\, \otimes$.

Exercise: generalise the construction above to an arbitrary category $\mathcal{A}$ instead of $\mathcal{S}et$. That is, given an arbitrary category $\mathcal{A}$, define the (pre)category $\mathcal{A}lg_{\mathcal{A}}(\mathit{I\!I})$ analogous to $\mathcal{A}lg(\mathit{I\!I})$ above. Also, generalise $\mathit{I\!I}$ to an arbitrary functor F.

The name '$\mathcal{A}lg$' is mnemonic for 'algebra' and derives from the following observation. The $\mathit{I\!I}$-ary operations are, in fact, very simple algebras. Conventionally, an **algebra** with a single operation $\oplus\colon I\!I A \to A$ is the pair $(A;\oplus)$, and A is called the **carrier**. Thanks to axiom unique-Type the carrier is fully determined by the operation itself, so that the operation itself can be considered the algebra.

1.23 Category $\mathcal{M}on$. Now that we have defined category $\mathcal{A}lg(I\!I)$, we take the opportunity to present another category, to be called $\mathcal{M}on$ (mnemonic for 'monoid'). It will be used in Section 1d below.

First recall the notion of monoid. A **monoid operation** is: a binary operation that is associative and has a neutral element (sometimes called unit, or even identity). Conventionally, a **monoid** is: a triple $(A;\oplus,e)$, where $\oplus\colon I\!I A \to A$ is a monoid operation and e is its neutral element. The carrier A is uniquely determined by $\oplus$ (thanks to axiom unique-Type, $A = \mathrm{tgt}(\oplus)$). Also, the neutral element for $\oplus$ is unique, since $e = e \oplus e' = e'$ for any two neutral elements e and e'. So, we might say that $\oplus$ alone is, or represents, the monoid. Anyway, we shall talk of monoid operations, rather than of monoids.

The significance of monoid operations for algorithmics is that the reduce-with-$\oplus$ is a well-defined function of type $Seq\, A \to A$ when $\oplus$ is a monoid operation; see paragraph 0.4.

Category $\mathcal{M}on$ is: the subcategory of $\mathcal{A}lg(I\!I)$ whose objects are the monoid operations, and whose morphisms are those f for which $f\colon \oplus \to_{I\!I} \otimes$ and $f(e) = e'$ where e, e' are the neutral elements of $\oplus, \otimes$.
Exercise: give an explicit definition of the objects, morphisms, typing, composition, and identities in $\mathcal{M}on$, and prove that the category axioms are fulfilled.
Exercise: prove that, in $\mathcal{S}et$, $\mathcal{M}on$ is not a full subcategory of $\mathcal{A}lg(I\!I)$.

1.24 More functors, new categories. Up to now we have seen only endofunctors, namely $I\!I$ and Seq; that is, functors whose source and target are equal. There are several reasons why it is useful to allow the source and target category of a functor to differ from each other. We briefly mention three of such reasons here. At the same time, these reasons demonstrate the need for building new categories out of given ones.

First, there is no problem in defining a notion of a 2-place functor, also called a **bifunctor**. (Exercise: try to define the notion of bifunctor formally; how would the bifunctor axioms read?) However, by a suitable definition of the *product* of two categories (like the cartesian product of sets), a 2-place functor on category $\mathcal{A}$ is just a normal functor $F\colon \mathcal{A} \times \mathcal{A} \to \mathcal{A}$. (Exercise: try to define the notion of the product category $\mathcal{A} \times \mathcal{B}$ of two categories $\mathcal{A}$ and $\mathcal{B}$. What are the objects, morphisms, typing, composition, and identities? Prove that these satisfy the category axioms.)

Second, let $\mathcal{A}$ be an arbitrary category, and consider the following mapping F from $\mathcal{A}$ to $\mathcal{S}et$.

$$\begin{array}{rcll} FA & = & \{g \mid g \text{ is a morphism in } \mathcal{A} \text{ with } \mathrm{src}\, g = A\}, & \text{an object in } \mathcal{S}et \\ Ff & = & g \ \mapsto\ f \,;_{\mathcal{A}}\, g & \\ & : & FA \to_{\mathcal{S}et} FB & \text{whenever } f\colon B \to_{\mathcal{A}} A\,. \end{array}$$

In view of the equation for Ff we might call Ff: 'precede with f', and we might write Ff alternatively as $(f;)$ or $(\circ f)$. Mapping F is like a functor; it has the properties that

$$\begin{array}{lcl} f\colon B \to_{\mathcal{A}} A & \Rightarrow & Ff\colon FA \to_{\mathcal{S}et} FB \\ F id & = & id \\ F(g \mathbin{;} f) & = & Ff \mathbin{;_{\mathcal{S}et}} Fg\,. \end{array}$$

Notice that in the left hand side A and B, and also f and g, are at the wrong place for F to be a functor. There is no problem in defining a notion of a **contravariant** functor, so that F is a contravariant functor. (Exercise: try to define the notion of a contravariant functor; how would the functor axioms read?) However, by a suitable definition of the *opposite* $\mathcal{A}^{op}$ of a category $\mathcal{A}$, mapping F is just a normal functor $F\colon \mathcal{A}^{op} \to \mathcal{S}et$. Category $\mathcal{A}^{op}$ is obtained from $\mathcal{A}$ by taking the objects, morphisms and identities from $\mathcal{A}$, and defining the typing and composition as follows:

$$\begin{array}{lcl} f\colon A \to_{\mathcal{A}^{op}} B & \equiv & f\colon B \to_{\mathcal{A}} A \\ f \mathbin{;_{\mathcal{A}^{op}}} g & = & g \mathbin{;_{\mathcal{A}}} f\,. \end{array}$$

One says that $\mathcal{A}^{op}$ is obtained from $\mathcal{A}$ by "reversing all arrows". (Exercise: verify that $\mathcal{A}^{op}$ is a category indeed. Verify also that mapping F above is a functor $F\colon \mathcal{A}^{op} \to \mathcal{S}et$.)

Third, sometimes we need to speak about, say, pairs of morphisms in $\mathcal{B}$ with a common source. (For example, the two extraction functions from a cartesian product form such a pair.) We can do this categorically as follows. Let $\mathcal{A}$ be the category determined by the following graph:

$$\bullet \xleftarrow{f} \bullet \xrightarrow{g} \bullet$$

Then each functor $F\colon \mathcal{A} \to \mathcal{B}$ determines a pair (Ff, Fg) of morphisms in $\mathcal{B}$ with a common source; and each such pair (f', g') in $\mathcal{B}$ can so be obtained by defining a suitable functor $F'\colon \mathcal{A} \to \mathcal{B}$. Moreover, such pairs and such functors determine each other uniquely. So, those pairs in $\mathcal{B}$ are in a precise sense equivalent to functors of type $\mathcal{A} \to \mathcal{B}$, where $\mathcal{A}$ is as above. (Exercise: how can you similarly express triples of morphisms with a common target? And what about triples (f, g, h) that satisfy $f \mathbin{;} g = h$?)

These three examples not only illustrate the usefulness of allowing a functor to have a different source and target, but also demonstrate the usefulness of defining new categories out of given ones, such as the product of categories, the opposite of a category, and several finite categories.

1.25 Composite functors. For functors $F\colon \mathcal{A} \to \mathcal{B}$ and $G\colon \mathcal{B} \to \mathcal{C}$ we define the mappings $I_{\mathcal{A}}$ and GF by

$$\begin{array}{lcl} I_{\mathcal{A}}\, x & = & x \\ (GF)x & = & G(Fx) \end{array}$$

for all objects and morphisms x in $\mathcal{A}$. In view of the defining equation we can write GFx without semantic ambiguity. We also write just I instead of $I_\mathcal{A}$ if $\mathcal{A}$ is irrelevant or clear from the context. Thus defined, I and GF are functors:

$$\begin{array}{l} I_\mathcal{A}\colon \mathcal{A} \to \mathcal{A} \\ F\colon \mathcal{A} \to \mathcal{B} \text{ and } G\colon \mathcal{B} \to \mathcal{C} \quad \Rightarrow \quad GF\colon \mathcal{A} \to \mathcal{C}\,. \end{array}$$

The properties ftr-Type and Functor are easily verified. (Exercise: do this.) Other important properties of these functors are: associativity of functor composition and neutrality of I with respect to functor composition:

$$\begin{array}{lllll} H\,(G\,F) & = & (H\,G)\,F & & \\ F\,I_\mathcal{A} & = & F & = & I_\mathcal{B}\,F \qquad \text{for } F\colon \mathcal{A} \to \mathcal{B}\,. \end{array}$$

The associativity implies that writing $H\,G\,F$ without parentheses causes no semantic ambiguity.
(Exercise: are $\mathit{Seq}\,\mathit{I\!I}$ and $\mathit{I\!I}\,\mathit{Seq}$ well defined, and if so, what structure do they represent? And what about $\mathit{Seq}\,\mathit{Seq}$?)

1.26 Category $\mathcal{C}at$. The above properties of functors suggest a (pre)category, called $\mathcal{C}at$. Take as objects all categories, as morphisms all functors, as typing the functor typing, as identity on $\mathcal{A}$ the identity functor $I_\mathcal{A}$, and as composition the functor composition. As usual, we can make a category of $\mathcal{C}at$, see paragraph 1.8. Thus our talking of 'type' of functors is justified.

However, there is a foundational problem lurking here. Is this new category $\mathcal{C}at$ an object in itself? An answer, be it yes or no, would give similar problems as the supposition that the set of all sets exists. We will neither use the new category in a formal reasoning, nor discuss ways out of this paradox.

1c Naturality

A natural transformation is nothing but a structure preserving map between functors. 'Structure preservation' makes sense, here, since we've seen already that a functor is, or represents, a structure that objects might have. We shall first give an example, and then present the formal definition.

1.27 Naturality in $\mathcal{S}et$. Let $F, G\colon \mathcal{A} \to \mathcal{S}et$ be functors. In the terminology of paragraph 1.21 each FA denotes a structured set and F denotes the structure itself. For example, $\mathit{I\!I}$ is the structure of pairs, Seq the structure of sequences, $\mathit{I\!I}\,\mathit{Seq}$ the structure of pairs of sequences, $\mathit{Seq}\,\mathit{Seq}$ the structure of sequences of sequences, and so on. A 'transformation' from structure F to structure G is: a family t of functions $t_A\colon FA \to GA$, mapping set FA to set GA for each A. A transformation t is 'natural' if: each t_A doesn't affect the constituents of the structured elements in FA but only reshapes the structure of the elements, from an F-structure into a G-structure; in other words,

reshaping the structure by means of t commutes with *subjecting the constituents* to an arbitrary morphism:

that is, $Ff \mathbin{;} t_{A'} = t_A \mathbin{;} Gf$ for all $f\colon\ A \to_{\mathcal{A}} A'$.

As an example, consider the functions $join_A\colon\ I\!I\,Seq\,A \to Seq\,A$ in $\mathcal{S}et$. Family $join$ is a natural transformation from $I\!I\,Seq$ to Seq, since

$$I\!I\,Seq\,f \mathbin{;} join_{A'} \quad = \quad join_A \mathbin{;} Seq\,f \qquad \text{for each } f\colon\ A \to A',$$

as you can easily verify. Transformation $join$ reshapes each $I\!I\,Seq$-structure into a Seq-structure, and doesn't affect the constituents of the elements in the structure.

In the next paragraph, naturality in general is defined like naturality in $\mathcal{S}et$; we abstract from $\mathcal{S}et$ and replace it by an arbitrary category $\mathcal{B}$. The formulas remain the same, but the interpretation above (in terms of functions, sets, and elements) may change.

1.28 Definition. Let $\mathcal{A}, \mathcal{B}$ be categories, and $F, G\colon\ \mathcal{A} \to \mathcal{B}$ be functors. A **transformation** in $\mathcal{B}$ from F to G is: a family ε of morphisms

1.29 $\qquad \varepsilon_A \quad : \quad FA \to_{\mathcal{B}} GA \qquad \text{for each } A \text{ in } \mathcal{A}.$ ntrf-Type

A transformation ε in $\mathcal{B}$ from F to G is **natural**, denoted $\varepsilon\colon\ F \dot{\to} G$ or $\varepsilon\colon\ F \dot{\to}_{\mathcal{B}} G$, if:

1.30 $\qquad Ff \mathbin{;} \varepsilon_B \quad = \quad \varepsilon_A \mathbin{;} Gf \qquad \text{for each } f\colon\ A \to_{\mathcal{A}} B.$ Ntrf

This formula is (so natural that it is) easy to remember: morphism $\varepsilon_{\text{target}\,f}$ has type $F(\text{target } f) \to G(\text{target } f)$ and therefore occurs at the target side of an occurrence of f; similarly $\varepsilon_{\text{source}\,f}$ occurs at the source side of an f. Moreover, since ε is a transformation *from* F *to* G, functor F occurs at the source side of an ε and functor G at the target side.

The notation εA is an alternative for ε_A, and uses ε as a function. We also say that ε is *natural* in its parameter. By default, $\gamma, \delta, \varepsilon, \eta, \kappa$ range over natural transformations. Exercise: prove that 1.29 follows from the assumption that 1.30 is well-typed. (So you need only remember 1.30.)

1.31 Example. Natural transformations are all over the place; we give here just two simple examples, and in paragraph 1.38 one application. The category under discussion is $\mathcal{S}et$.

First, consider the transformation rev that yields the reversal of its argument:

$$rev_A \quad : \quad Seq\,A \to Seq\,A$$

$$rev_A \quad = \quad [a_0, \ldots, a_{n-1}] \mapsto [a_{n-1}, \ldots, a_0] \, .$$

Thus, *rev* reshapes a *Seq*-structure into a *Seq*-structure, not affecting the constituents of its arguments. Family *rev* is a natural transformation typed

$$rev \quad : \quad Seq \dot{\to} Seq \, ,$$

since for all $f\colon\ A \to B$

$$Seq\, f \,;\, rev_B \quad = \quad rev_A \,;\, Seq\, f \, ,$$

as is easily verified.

Second, consider the transformation *inits* that yields all initial parts of its argument:

$$\begin{array}{lcl} inits_A & : & Seq\, A \to Seq\, Seq\, A \\ inits_A & = & [a_0, \ldots, a_{n-1}] \mapsto [[\,], [a_0], \ldots, [a_0, \ldots, a_{n-1}]] \, . \end{array}$$

Thus, *inits* reshapes a *Seq*-structure into a *Seq Seq*-structure, not affecting the constituents of its arguments. Family *inits* is a natural transformation typed

$$inits \quad : \quad Seq \dot{\to} Seq\, Seq \, ,$$

since for all $f\colon\ A \to B$

$$Seq\, f \,;\, inits_B \quad = \quad inits_A \,;\, Seq\, Seq\, f \, ,$$

as is easily verified.

Exercise: verify that each of the following well-known operations is a natural transformation of the given type:

$$\begin{array}{lcll} tip & : & I \dot{\to} Seq & \\ concat & : & Seq\, Seq \dot{\to} Seq & \text{equals } +\!\!\!+/, \quad \text{also called } \mathit{flatten} \\ segs & : & Seq \dot{\to} Seq\, Seq & \\ parts & : & Seq \dot{\to} Seq\, Seq\, Seq & \text{yields all } \mathit{part}\text{itions of its argument} \\ take\, n & : & Seq \dot{\to} Seq & \\ zip & : & I\!I\, Seq \dot{\to} Seq\, I\!I & \\ rotate & : & Seq \dot{\to} Seq & \\ swap & : & I\!I \dot{\to} I\!I & \mathit{swap}\text{s the components of its argument} \\ exl & : & I\!I \dot{\to} I & \text{extracts the } \mathit{l}\text{eft component of a pair}\, . \end{array}$$

We shall later see how to formulate the naturality of $\_ : \_$ and *nil*, and of *take* (not fixing one of its arguments), and how to formulate a more general naturality of *swap* and *exl* (not restricting their arguments to the same type), and that reduce itself, operation $\_/$, is a natural transformation in category $\mathcal{M}on$.

1.32 Composition of natural transformations. For functors F, G, H, J, K and natural transformations $\varepsilon\colon F \dot{\to} G$ and $\eta\colon G \dot{\to} H$ we define transformations id_F, $\varepsilon ; \eta$, $J\varepsilon$, and ε_K by

$$\begin{array}{lcl} (id_F)_A & = & id_{(FA)} \\ (\varepsilon ; \eta)_A & = & \varepsilon_A ; \eta_A \\ (J\varepsilon)_A & = & J(\varepsilon_A) \\ (\varepsilon_K)_A & = & \varepsilon_{KA} . \end{array}$$

We shall write $id_{F\,A}$ and $J\varepsilon_A$ and $\varepsilon_{K\,A}$ without parentheses; in view of the equations this causes no semantic ambiguity. An alternative notation for ε_K is εK; so $(\varepsilon K)A = \varepsilon(KA)$ and we then write εKA without parentheses too. Similarly, $(J\varepsilon)K = J(\varepsilon K)$ and we write simply $J\varepsilon K$. These transformations are natural:

1.33 $\quad id_F\colon F \dot{\to} F$ $\quad$ ntrf-Id

1.34 $\quad \varepsilon\colon F \dot{\to} G$ and $\eta\colon G \dot{\to} H \Rightarrow \varepsilon ; \eta\colon F \dot{\to} H$ $\quad$ ntrf-Compose

1.35 $\quad \varepsilon\colon F \dot{\to} G \Rightarrow J\varepsilon\colon JF \dot{\to} JG$ $\quad$ ntrf-Ftr

1.36 $\quad \varepsilon\colon F \dot{\to} G \Rightarrow \varepsilon_K\colon FK \dot{\to} GK$ $\quad$ ntrf-Poly

Notice that for laws 1.35 and 1.36 to make sense, F and G have a common source and a common target, the source of J is the target of F and G, and the target of K is the source of F and G. The proofs are quite simple; we prove only law ntrf-Compose. As regards property ntrf-Type for $\varepsilon ; \eta$ we argue

$$\begin{array}{ll} & (\varepsilon ; \eta)_A\colon FA \to HA \\ \equiv & \qquad \text{definition of } \varepsilon ; \eta \\ & \varepsilon_A ; \eta_A\colon FA \to HA \\ \Leftarrow & \qquad \text{composition-Type} \\ & \varepsilon_A\colon FA \to GA \text{ and } \eta_A\colon GA \to HA \\ \Leftarrow & \qquad \text{definition } \dot{\to} \\ & \varepsilon\colon F \dot{\to} G \text{ and } \eta\colon G \dot{\to} H. \end{array}$$

And to show the naturality, property Ntrf, for $\varepsilon ; \eta$, we argue, for arbitrary $f\colon A \to B$,

$$\begin{array}{ll} & Ff ; (\varepsilon ; \eta)_B = (\varepsilon ; \eta)_A ; Hf \\ \equiv & \qquad \text{definition } (\varepsilon ; \eta) \\ & Ff ; \varepsilon_B ; \eta_B = \varepsilon_A ; \eta_A ; HF \\ \equiv & \qquad \text{premise: naturality } \varepsilon \text{ and } \eta \\ & \varepsilon_A ; Gf ; \eta_B = \varepsilon_A ; Gf ; \eta_B \\ \equiv & \qquad \text{equality} \\ & \mathit{true}. \end{array}$$

(Exercise: prove laws 1.33, 1.35, and 1.36.)

Further important properties of natural transformations are associativity of composition and neutrality of id_F with respect to composition of natural transformations:

$$\begin{array}{lllll} \varepsilon ; (\eta ; \theta) & = & (\varepsilon ; \eta) ; \theta & & \\ id_F ; \varepsilon & = & \varepsilon & = & \varepsilon ; id_G \end{array}$$

for $\varepsilon\colon F \dot{\to} G$. The proof of these properties is simple; the properties are inherited from composition and identities of the category. (Exercise: prove these claims.)

1.37 Category $\mathcal{F}tr(\mathcal{A}, \mathcal{B})$. The properties of composite natural transformations suggest a category. Let $\mathcal{A}$ and $\mathcal{B}$ be a category. Form a new category, commonly denoted $\mathcal{F}tr(\mathcal{A}, \mathcal{B})$, as follows. Take as objects all functors from $\mathcal{A}$ to $\mathcal{B}$, as morphisms all natural transformations (from functors with type $\mathcal{A} \to \mathcal{B}$ to functors with type $\mathcal{A} \to \mathcal{B}$), as typing the typing of naturality (above denoted $\dot{\to}$), as identities all identity natural transformations id_F, and as composition the composition of natural transformations defined above. Thus defined, $\mathcal{F}tr(\mathcal{A}, \mathcal{B})$ is a pre-category and even a category. (Exercise: verify this.)

1.38 Application. Continuing the example of paragraph 1.31, we define a family *tails* as follows. Function $tails_A$ yields all tail parts of its argument sequence as its result:

$$\begin{array}{lll} tails_A & : & Seq\,A \to Seq\,Seq\,A \\ tails_A & = & rev_A ; inits_A ; Seq\,rev_A\,. \end{array}$$

One may now suspect that, for all $f\colon A \to B$,

$$Seq\,f ; tails_B \quad = \quad tails_A ; Seq\,Seq\,f\,,$$

so that $tails\colon Seq \dot{\to} Seq\,Seq$. Indeed, this is almost immediate by the laws given in the previous paragraph:

$$\begin{array}{ll} & tails\colon Seq \dot{\to} Seq\,Seq \\ \equiv & \quad \text{definition } tails \\ & rev ; inits ; Seq\,rev\colon Seq \dot{\to} Seq\,Seq \\ \Leftarrow & \quad \text{ntrf-Compose} \\ & rev\colon Seq \dot{\to} Seq, \quad inits\colon Seq \dot{\to} Seq\,Seq, \quad Seq\,rev\colon Seq\,Seq \dot{\to} Seq\,Seq \\ \Leftarrow & \quad \text{ntrf-Ftr, premises: naturality } rev \text{ and } inits \\ & true. \end{array}$$

In effect, the proof of this semantic property is nothing but type checking (viewing "$\_ : \_ \dot{\to} \_$" as a typing, and nowhere using the actual meaning of *inits*, *rev*, and *tails*). Hadn't we had available the concept and properties of naturality, the proof would have been much longer. Indeed, explicitly using the equalities

$$\begin{array}{lcl} Seq\,f\,;\,rev_B & = & rev_A\,;\,Seq\,f \\ Seq\,f\,;\,inits_B & = & inits_A\,;\,Seq\,Seq\,f \end{array}$$

for all $f\colon A \to B$, the proof of $Seq\,f\,;\,tails_B = tails_A\,;\,Seq\,Seq\,f$ would run as follows.

$$\begin{array}{ll} & Seq\,f\,;\,tails_B \\ = & \qquad \text{definition } tails \\ & Seq\,f\,;\,rev_B\,;\,inits_B\,;\,Seq\,rev_B \\ = & \qquad \text{equation for } rev \\ & rev_A\,;\,Seq\,f\,;\,inits_B\,;\,Seq\,rev_B \\ = & \qquad \text{equation for } inits \\ & rev_A\,;\,inits_A\,;\,Seq\,Seq\,f\,;\,Seq\,rev_B \\ = & \qquad \text{Functor for } Seq \\ & rev_A\,;\,inits_A\,;\,Seq(Seq\,f\,;\,rev_B) \\ = & \qquad \text{equation for } rev \\ & rev_A\,;\,inits_A\,;\,Seq(rev_A\,;\,Seq\,f) \\ = & \qquad \text{Functor for } Seq \\ & rev_A\,;\,inits_A\,;\,Seq\,rev_A\,;\,Seq\,Seq\,f \\ = & \qquad \text{definition } tails \\ & tails_A\,;\,Seq\,Seq\,f. \end{array}$$

1.39 Omitting subscripts. For readability we shall often omit the subscripts or arguments to natural transformations when they can be retrieved from contextual information. Here is an example; you are not supposed to understand the 'meaning' of the formulas.

Let the following be given:

$$\begin{array}{lcl} F & : & \mathcal{A} \to \mathcal{B} \\ G & : & \mathcal{B} \to \mathcal{A} \\ \eta & : & I_{\mathcal{A}} \dot{\to} GF \\ \varepsilon & : & FG \dot{\to} I_{\mathcal{B}}\,, \end{array}$$

and consider formula

$$\eta\,;\,G\varepsilon \quad = \quad id\,.$$

The following procedure gives the most general subscripts that make the formula well typed. Use $a, b, c, \ldots$ as type variables (the "unknows"), use these as the subscripts, and write the source and target type within braces at the source and target side of the morphisms, thus:

$$\{a\}\quad \eta_b \ \{c\} \ ; \ \{d\} \quad G(\ \{e\} \quad \varepsilon_f \ \{g\}\ )\ \{h\} \quad = \quad \{j\} \quad id_k \ \{l\}\ .$$

The typing axioms generate a collection of equations for the type variables:

typing η :	$a, c = b, GFb$	on account of ntrf-Type
typing $;$:	$c = d$	on account of composition-Type
typing $G\varepsilon$:	$d, h = Ge, Gg$	on account of ftr-Type
typing ε :	$e, g = FGf, f$	on account of ntrf-Type
typing id :	$j = k = l$	on account of identity-Type
typing $=$:	$a, h = j, l$.	

A most general (least constraining) solution for this collection of equations can be found by the unification algorithm, and yields

$$a = b = h = j = l = k = Gf$$
$$c = d = GFGf$$
$$e = FGf$$
$$g = f\ .$$

Hence, writing B for type variable f, and filling in the subscripts, the formula reads: for arbitrary object B in $\mathcal{B}$,

$$\eta_{GB} \ ; G\varepsilon_B \quad = \quad id_{GB} \quad : \quad GB \to_{\mathcal{A}} GB\ ,$$

or, writing the subscripts as arguments, and abstracting from B,

$$\eta G \ ; G\varepsilon \quad = \quad idG \quad : \quad G \dot{\to} G\ .$$

Exercise: infer in a similar way the categories, and the typing of the functors in:

$$\eta \quad : \quad I \dot{\to} GF$$
$$\varepsilon \quad : \quad FG \dot{\to} I\ .$$

Exercise: assuming

$$\varepsilon, \eta \quad : \quad F \dot{\to} F\,F$$
$$\kappa \quad : \quad F\,F \dot{\to} F\ ,$$

find the most general subscripts that make $\varepsilon \ ; \ F\eta \ ; \ \kappa$ a well-typed term denoting a morphism. (What function does the term denote if the category is $\mathcal{S}et$, $F = Seq$, and $\varepsilon, \eta, \kappa = inits, tails, join/$?)

1d Adjunctions

An adjunction is a particular one-one correspondence between, on the one hand, the morphisms of a certain type in one category, and, on the other hand, the morphisms of a certain type in another category. The correspondence can be formulated as an equivalence between two equations (in the two categories, respectively). An adjunction has many properties, and many different but equivalent definitions.

1.40 Example. Here is a law for sequences; it has a lot of well-known consequences, as we shall show in a while.

> "Each homomorphism on sequences is uniquely determined (as a 'map' followed by a reduce) by its restriction to the singleton sequences."

To be precise, the law reads as follows.

> Let A be an arbitrary set, and $\otimes$ be an arbitrary monoid operation, say with target set B. Then, for all $f\colon A \to B$ and all $g\colon (+\!\!+_A) \to_{\mathcal{M}on} \otimes$,
>
> $$f = tip_A \,;\, g \qquad \equiv \qquad Seq\, f \,;\, \otimes/ = g\,. \qquad \text{SeqAdj}$$

Thus we may call f the 'restriction of g to the tip elements' and write $f = \llbracket g \rrbracket_{A,\otimes} = tip_A \,;\, g$. Also, we may call g the 'extension of f to a homomorphism from $(+\!\!+_A)$ to $\otimes$' and write $g = \lceil\!\lceil f \rceil\!\rceil_{A,\otimes} = Seq\, f \,;\, \otimes/$. With these definitions, and omitting the subscripts, the equivalence reads:

$$f = \llbracket g \rrbracket \qquad \equiv \qquad \lceil\!\lceil f \rceil\!\rceil = g\,.$$

This equivalence expresses that $\llbracket\ \rrbracket$ and $\lceil\!\lceil\ \rceil\!\rceil$ are each other's inverse, and constitute a one-one correspondence between functions (of a certain type) and homomorphisms (of a certain type). Mappings $\llbracket\ \rrbracket$ and $\lceil\!\lceil\ \rceil\!\rceil$ are called **lad** and **rad**, respectively, from *l*eft *ad*jungate and *r*ight *ad*jungate; these names and notations are not standard in category theory.

The above law is an (almost full-fledged) instance of an adjunction. The significance for algorithmics may be evident from the consequences of SeqAdj listed in paragraph 1.49.

1.41 Definition. An adjunction involves several data:

- Two categories $\mathcal{A}$ and $\mathcal{B}$.
  [In the above example $\mathcal{A} = \mathcal{S}et$ and $\mathcal{B} = \mathcal{M}on$.]

- Two functors $F\colon \mathcal{A} \to \mathcal{B}$ and $G\colon \mathcal{B} \to \mathcal{A}$.
  [Above $Ff = Seq\, f$ and $Gg = g$ for morphisms f and g. (The fact that above G is the identity on morphisms and therefore is invisible in the left-hand equation, makes the example a bit special.) For objects the above functors act as follows: $FA = (+\!\!+_A)$, a monoid operation, and $G(\oplus)$ = the target set of $\oplus$. Exercise: check that, thus defined, F and G are functors.]

- Two transformations $\eta\colon I \to GF$ in $\mathcal{A}$ and $\varepsilon\colon FG \to I$ in $\mathcal{B}$.
  [Above $\eta = tip$ and $\varepsilon_\otimes = \otimes/$.]

An **adjunction** is: such a sextuple $\mathcal{A}, \mathcal{B}, F, G, \eta, \varepsilon$ satisfying the following property:

For arbitrary objects A in $\mathcal{A}$ and B in $\mathcal{B}$, and morphisms $f\colon A \to_{\mathcal{A}} GB$ and $g\colon FA \to_{\mathcal{B}} B$,

1.42 $\quad f = \eta_A \mathbin{;} Gg \quad \equiv \quad Ff \mathbin{;} \varepsilon_B = g\,.$ Adjunction

If, given $\mathcal{A}$, $\mathcal{B}$, F, G, there exist η, ε such that the sextuple forms an adjunction, then F is called **left adjoint** to G, and G **right adjoint** to F.
Exercise: verify that the law for sequences in paragraph 1.40 is an adjunction indeed, with $\mathcal{A}, \mathcal{B}, F, G, \eta, \varepsilon$ defined as suggested a few lines up.

1.43 Corollaries. An adjunction satisfies a lot of properties, some of which enable an alternative, equivalent definition. Here we mention just a few of the properties, postponing the (mostly simple) proofs to a later time. We list these properties only to give some idea of the richness and importance of the notion of adjunction.

Let $\mathcal{A}, \mathcal{B}, F, G, \eta, \varepsilon$ form an adjunction.

1. Then η and ε determine each other, and they are *natural* transformations $\eta\colon I \dot{\to} GF$ and $\varepsilon\colon FG \dot{\to} I$. This gives rise to two alternative characterisations of an adjunction, one not involving η and another not involving ε.
   Exercise: check the naturality of $\eta_A = tip_A$ in Set and of $\varepsilon_\otimes = \otimes/$ in $\mathcal{M}on$.

2. Define mappings $\llbracket\ \rrbracket_{\_,\_}$ and $\lceil\!\lceil\ \rceil\!\rceil_{\_,\_}$, sending morphisms (of a certain type) from $\mathcal{B}$ to $\mathcal{A}$, and from $\mathcal{A}$ to $\mathcal{B}$ respectively, by:

   1.44 $\quad \llbracket g \rrbracket_{A,B} \; = \; \eta_A \mathbin{;} Gg \quad : A \to_{\mathcal{A}} GB$ whenever $g\colon FA \to_{\mathcal{B}} B$ lad-Def

   1.45 $\quad \lceil\!\lceil f \rceil\!\rceil_{A,B} \; = \; Ff \mathbin{;} \varepsilon_B \quad : FA \to_{\mathcal{B}} B$ whenever $f\colon A \to_{\mathcal{A}} GB$ rad-Def

   Then, for f and g of the appropriate type, and omitting the subscripts,

   1.46 $\quad f = \llbracket g \rrbracket \quad \equiv \quad \lceil\!\lceil f \rceil\!\rceil = g\,,$ Inverse

   and $\llbracket\ \rrbracket$ and $\lceil\!\lceil\ \rceil\!\rceil$ satisfy the following *fusion* properties:

   1.47 $\quad \llbracket Fx \mathbin{;} g \mathbin{;} y \rrbracket \; = \; x \mathbin{;} \llbracket g \rrbracket \mathbin{;} Gy$ lad-Fusion

   1.48 $\quad \lceil\!\lceil x \mathbin{;} f \mathbin{;} Gy \rceil\!\rceil \; = \; Fx \mathbin{;} \lceil\!\lceil f \rceil\!\rceil \mathbin{;} y\,,$ rad-Fusion

   for $x\colon A' \to A$, $f\colon A \to GB$ in $\mathcal{A}$, and $g\colon FA \to B$, $y\colon B \to B'$ in $\mathcal{B}$.

3. From $\llbracket\ \rrbracket$ and $\lceil\!\lceil\ \rceil\!\rceil$ that satisfy 1.46, 1.47, and 1.48, natural transformations η and ε can be retrieved. This gives again another characterisation of an adjunction, in which $\llbracket\ \rrbracket$ and $\lceil\!\lceil\ \rceil\!\rceil$ do occur and η and ε don't.

4. In addition, such $\llbracket\ \rrbracket$ and $\lceil\!\lceil\ \rceil\!\rceil$ determine each other. This gives rise to yet another pair of characterisations of an adjunction; one in which $\llbracket\ \rrbracket$ doesn't occur, and one in which $\lceil\!\lceil\ \rceil\!\rceil$ doesn't occur.

1.49 Example continued. Here are some consequences of the adjunction in mentioned in paragraph 1.40. Actually, all these properties are instantiations of the corollaries mentioned in paragraph 1.43. So, these properties can be proved from the adjunction property alone, without referring to the actual meaning of tip, $\_/$, Seq, and the very notion of 'sequences'.

$$\begin{array}{lcll}
tip \mathbin{;} Seq\, f & = & f \mathbin{;} tip & \\
Seq\, g \mathbin{;} \otimes/ & = & \oplus/ \mathbin{;} g & \text{whenever } g\colon \oplus \to_{I\!I} \otimes \\
tip \mathbin{;} Seq\, f \mathbin{;} \oplus/ & = & f & \\
Seq(tip \mathbin{;} g) \mathbin{;} \oplus/ & = & g & \text{whenever } g\colon (+\!\!+) \to_{I\!I} \oplus \\
tip \mathbin{;} \oplus/ & = & id & \\
Seq\, tip \mathbin{;} +\!\!+/ & = & id\,. &
\end{array}$$

Exercise: derive these properties from the adjunction property. Take care not to use the actual meaning of tip, $\_/$, and Seq.
Exercise: try to give some subcollections of this list of properties that are equivalent to the adjunction property.
Exercise: try to formulate these properties in terms of $\mathcal{A}, \mathcal{B}, F, G, \eta, \varepsilon, \llbracket\ \rrbracket, \lceil\!\lceil\ \rceil\!\rceil$, and try to derive them from the adjunction property. (This will be done for you in a later section.)

1.50 More corollaries. Here are some more corollaries. Again we list them here only to show the importance of the concept. These corollaries may be harder to understand than those in paragraph 1.43, due to the higher level of abstraction.

1. Adjoint functors determine each other "up to isomorphism". We shall explain the concept of isomorphism later. More precisely, if $\mathcal{A}, \mathcal{B}, G$ can be completed to an adjunction $\mathcal{A}, \mathcal{B}, F, G, \eta, \varepsilon$, then F is unique up to isomorphism.

   As a consequence, the existence of some F', η', ε' for which the sextuple $\mathcal{S}et$, $\mathcal{M}on$, F', G, η', ε' (with G as in the above example) forms an adjunction, *is equivalent to* the existence of a monoid operation $+\!\!+'_A$ $(= F'A)$ that has the categorical properties of 'the monoid operation of sequences'. Thus, a datatype like that of sequences can be defined by a certain adjunction.
   Exercise: suppose there exist Seq', tip', $\_/'$, and $+\!\!+'_A$ that, substituted for Seq, tip, $\_/$, and $+\!\!+_A$, make the adjunction property in paragraph 1.40 well-typed and true. Convince yourself (informally) that $(\mathrm{tgt}(+\!\!+'_A);\ +\!\!+'_A,\ tip'_A,$ the neutral element of $+\!\!+'_A)$ might be called 'the datatype of sequences'.

2. (Surely, it'll take some time and exercising before you can easily grasp the following highly abstract statement.) The fusion properties of $\llbracket\ \rrbracket$ and $\lceil\!\lceil\ \rceil\!\rceil$ are equivalent to the statement that $\llbracket\ \rrbracket$ and $\lceil\!\lceil\ \rceil\!\rceil$ are morphisms of a certain type in category $\mathcal{F}tr(\mathcal{A}^{op} \times \mathcal{B},\ \mathcal{S}et)$ (where the objects are functors and the morphisms are natural transformations, see paragraph 1.37). So, $\llbracket\ \rrbracket$ and $\lceil\!\lceil\ \rceil\!\rceil$ are natural transformations

"of a higher type", and the omission of the subscripts to $⦇\,⦈$ and $[\![\,]\!]$ thus falls under our convention for natural transformations.

3. If F is left adjoint to G, or, equivalently, G is right adjoint to F, then F preserves colimits (such as initial objects and sums; all these notions will be defined later), and G preserves limits (such as final objects and products, again to be defined later).

1.51 More on adjointness. In Appendix A we give formalisations of (most of) the above claims, as well as their formal proofs. With the exception of the part 'Initiality and colimit as adjointness', that text uses no other concepts than those known here, so that you may start reading it right now. It is an excellent demonstration of the calculational approach to category theory.

1e Duality

Dualisation is a formal manipulation with practical significance. For example, the set-theoretic notions of cartesian product and disjoint union are characterised categorically by notions that are each other's dual. As another example, the categorical characterisation of a 'datatype for which functions can be defined by induction on the structure of the argument' (like the datatype of sequences) is dual to the categorical characterisation of a 'datatype for which functions can be defined by induction on the structure of their result' (like the datatype of infinite lists, or streams). Dualisation also applies to theorems and proofs, thus cutting work in half.

1.52 Definition. The dual of a term in the categorical language is defined by:

$$
\begin{array}{lcll}
\mathit{dual}\,A & = & A & \text{for object term } A \\
\mathit{dual}\,x & = & x & \text{for morphism } \textit{variable } x \\
\mathit{dual}(f\colon A \to B) & = & \mathit{dual}\,f\colon B \to A & \text{(note the swap of } A \text{ and } B\text{)} \\
\mathit{dual}(f \mathbin{;} g) & = & \mathit{dual}\,g \mathbin{;} \mathit{dual}\,f & \text{(note the swap of } f \text{ and } g\text{)} \\
\mathit{dual}(\mathit{id}_A) & = & \mathit{id}_A\,. &
\end{array}
$$

Clearly, dualising is its own inverse, that is, $\mathit{dual}(\mathit{dual}\,t) = t$ for each term t. Another easy way of dualising a morphism term is simply replacing each $;$ by $\circ$. However, the presence of both compositions for the same morphisms is not practical. As an example, the following two statements are each other's dual.

1.53 $\quad \forall B\ \exists! f :: \quad f\colon A \to B$

1.54 $\quad \forall B\ \exists! f :: \quad f\colon B \to A\,.$

Dualising a less trivial statement may be more instructive. Here is one; don't try to understand what it means, we'll meet it in the sequel. Apart from dualising the statement, we also rename some bound variables and interchange the sides of the left-hand equation (which doesn't affect the *meaning*).

$$\begin{array}{lllll} \exists (\!|\_|\!) & \forall B \;\; \forall f\colon A \to B \;\; \forall \varphi\colon FB \to B :: & \alpha \, ; f = Ff \, ; \varphi & \equiv & f = (\!|\varphi|\!) \\ \exists [\![\_]\!] & \forall B \;\; \forall g\colon B \to A \;\; \forall \psi\colon B \to FB :: & \psi \, ; Fg = g \, ; \alpha & \equiv & g = [\![\psi]\!] . \end{array}$$

Exercise: infer the typing of F, α, $(\!|\_|\!)$, and $[\![\_]\!]$ in these formulas. Notice that the type of the free variable α changes due to the dualisation.

1.55 Corollary. For each definition expressed in the categorical language, of a concept or construction *xxx*, you obtain another concept, often called *co-xxx* if no better name suggests itself, by dualising each term in the definition. For example, an object A is *initial* in a category if: formula 1.53 holds for A. (In $\mathcal{S}et$ the only initial object is the empty set.) Dually, an object A is *co-initial*, conventionally called *final* or *terminal*, if: formula 1.54 holds for A. (In $\mathcal{S}et$ the final objects are precisely the singleton sets.) Similarly, the other two formulas above define dual notions of α.

Also, for each equation $f = g$ provable from the axioms of category theory (hence valid for all categories), the equation $\mathit{dual}\, f = \mathit{dual}\, g$ is provable too. (Exercise: check this for the axioms of a category.) Thus dualisation cuts work in half, and gives each time two concepts or theorems for the price of one.

1.56 Examples. We shall meet many examples in the sequel, notably the examples mentioned in the introduction to this section.

Let it it suffice here to say that the opposite category $\mathcal{A}^{op}$ (defined in paragraph 1.24) is obtained by dualising each notion of $\mathcal{A}$, that is,

$$\begin{array}{lcl} \text{an object in } \mathcal{A}^{op} & \text{is:} & \mathit{dual}\, A \text{ for some object } A \text{ in } \mathcal{A} \\ \text{a morphism in } \mathcal{A}^{op} & \text{is:} & \mathit{dual}\, f \text{ for some morphism } f \text{ in } \mathcal{A} \\ f\colon A \to B \text{ in } \mathcal{A}^{op} & \equiv & \mathit{dual}(f\colon A \to B) \text{ in } \mathcal{A} \\ f \,;_{\mathcal{A}^{op}} g & = & \mathit{dual}(f \,;_{\mathcal{A}} g) \\ id_{\mathcal{A}^{op},A} & = & \mathit{dual}(id_{\mathcal{A},A}) . \end{array}$$

It follows that $(\mathcal{A}^{op})^{op} = \mathcal{A}$, and that the dual of a statement holds for $\mathcal{A}$ if and only if the statement itself holds for $\mathcal{A}^{op}$. (So again, if a statement is true for all categories, then its dual is true for all categories too.)

Exercise: prove that $F\colon \mathcal{A} \to \mathcal{B}$ equivales $F\colon \mathcal{A}^{op} \to \mathcal{B}^{op}$.
Exercise: dualise the notion of natural transformation.
Exercise: dualise the notion of adjunction, and of 'being a left adjoint'.

Chapter 2

Constructions in categories

In this chapter we discuss some categorical concepts by which some familiar (set-theoretic or other) concepts can be expressed in categorical terms. It turns out that most characterisations do not fix the objects and morphisms exactly, but only 'up to isomorphism'. Isomorphic objects are essentially the same, as regards the "observations" by the morphisms of the category.

There is a general pattern in several definitions; they turn out to define an *initial* or *final* object in a category built upon the category of interest ("the universe of discourse"). Therefore we shall discuss initiality and finality extensively before we turn to the other concepts.

2.1 Default category. The declaration that a category is the **default category** means that it is this category, rather than another one, that should be mentioned whenever there is an ambiguity. For example, when $\mathcal{A}$ is declared the default category, and several other (auxiliary) categories are discussed in the same context (in particular categories built upon $\mathcal{A}$), then a formula like $f\colon A \to B$ really means $f\colon A \to_{\mathcal{A}} B$, and 'an object' really means 'an object in $\mathcal{A}$'.

2a Iso, epic, and monic

All of the following definitions are relative to a category, the default one, which we don't mention explicitly to simplify the formulas. As usual, each formula is understood to be universally quantified with "for all $\langle$*variables not mentioned in the context*$\rangle$ of the appropriate type". Appropriateness of the type means that the formula is well-typed; see paragraph 1.2.

2.2 Definition. A **post-inverse** of a morphism f is: a morphism g such that

$$f \,;\, g = id\,.$$

A **pre-inverse** of a morphism f is: a morphism g such that

$$g \mathbin{;} f = id .$$

An **inverse** of a morphism f is: a morphism g that is a pre- and post-inverse of f:

$$f \mathbin{;} g = id \qquad \wedge \qquad g \mathbin{;} f = id .$$

A morphism f has at most one inverse (see below); it is denoted $f^{\cup}$ if it exists.
An **isomorphism** is: a morphism that has an inverse.
A morphism f is **epic** or an **epimorphism** if:

$$f \mathbin{;} x = f \mathbin{;} y \qquad \equiv \qquad x = y .$$

A morphism f is **monic** or an **monomorphism** if:

$$x \mathbin{;} f = y \mathbin{;} f \qquad \equiv \qquad x = y .$$

In both equivalences the $\Leftarrow$-part is an application of Leibniz.
Two objects are **isomorphic** if: there exists an isomorphism between them. If A and B are isomorphic, and $f\colon A \to B$ is an isomorphism, then we write $A \cong B$ and $f\colon A \cong B$, supplying a subscript $\mathcal{A}$ when appropriate.

2.3 Facts. In $\mathcal{S}et$, an isomorphism is a bijective function, a monomorphism is an injective function, and an epimorphism is a surjective function, and vice versa. (Exercise: prove this.) So, in $\mathcal{S}et$ a morphism is an isomorphism if and only if it is both monic and epic. This does not hold in general: in the category suggested by $\bullet \longrightarrow \bullet$ (containing three morphisms in total), the non-identity morphism is both epic and monic, and not an isomorphism.

A morphism has at most one inverse. For suppose that $f\colon A \to B$ has inverses $g, h\colon B \to A$. Then g and h are equal:

$$
\begin{array}{ll}
 & g = h \\
\equiv & \qquad \text{Identity on both sides} \\
 & id_B \mathbin{;} g = h \mathbin{;} id_A \\
\equiv & \qquad \text{lhs: } h \text{ is a pre-inverse of } f\text{, that is, } id_B = h \mathbin{;} f\text{,} \\
 & \qquad \text{rhs: } g \text{ is a post-inverse of } f\text{, that is, } id_A = f \mathbin{;} g \\
 & h \mathbin{;} f \mathbin{;} g = h \mathbin{;} f \mathbin{;} g \\
\equiv & \qquad \text{equality} \\
 & \mathit{true}.
\end{array}
$$

In fact, this calculation shows that a pre- and a post-inverse for the same morphism are equal.

It is not true that there is at most one isomorphism between a pair of objects: in $\mathcal{S}et$ all sets of cardinality n are isomorphic in $n!$ ways.

The notions of pre- and post-inverse, and of epi- and monomorphism are each other's dual. The notions of inverse, and of isomorphism, are their own dual.

Exercise: what does it mean for two sets to be isomorphic in $\mathcal{S}et$?
Exercise: what does it mean for two elements of a pre-ordered set to be isomorphic as objects in the category determined by the pre-ordered set (see paragraph 1.10)?
Exercise: spell out in terms of $\mathcal{S}et$ what it means for two $I\!I$-ary operations to be isomorphic objects in $\mathcal{A}lg(I\!I)$.
Exercise: spell out in terms of $\mathcal{B}$ what it means for two functors from $\mathcal{A}$ to $\mathcal{B}$ to be isomorphic as objects in $\mathcal{F}tr(\mathcal{A},\mathcal{B})$. (You may wonder whether this notion of isomorphic functors coincides with your intuitive, informal, notion of isomorphic 'structures', viewing a functor as a structure, as in paragraph 1.27.) Are functors $I\!I\,Seq$ and $Seq\,I\!I$ isomorphic in $\mathcal{F}tr(\mathcal{S}et,\mathcal{S}et)$?
Exercise: prove that the composition of isomorphisms is an isomorphism again. What is the inverse of a composite isomorphism?
Exercise: prove that each isomorphism is both monic and epic.
Exercise: given that $\mathcal{A}$ is a subcategory of $\mathcal{B}$, prove that each monomorphism in $\mathcal{B}$ is monic in $\mathcal{A}$.
Exercise: prove in $\mathcal{S}et$ that a function is epic iff it has a pre-inverse. It is not true that in each category a morphism is epic iff it has a pre-inverse. Similarly, in $\mathcal{S}et$ a morphism is injective iff it has a post-inverse, but this is not so in an arbitrary category.

2.4 "Up to isomorphism". The relation $\cong$ is an equivalence relation: reflexive, symmetric and transitive. Let P be a property of objects that holds for all objects of precisely one class of isomorphic objects. Then we sometimes speak of **the P-object**, meaning: an arbitrary but fixed object for which P holds. And we also say that the P-object is **unique up to isomorphism**. For example, in $\mathcal{S}et$ "the set with 17 elements" is unique up to isomorphism.

If a property P holds for precisely one class of isomorphic objects, and for any two objects in the class there is precisely one isomorphism from the one to the other, then we say that the P-object is **unique up to a unique isomorphism**. For example, in $\mathcal{S}et$ the one-point set is unique up to a unique isomorphism, and the two-point set is not.

2.5 Discussion. Isomorphic objects are often called 'abstractly the same' since for most categorical purposes one is as good as the other: each morphism to/from the one can be extended to a morphism to/from the other using the morphisms that establish the isomorphism. (The preceding sentence is informal intuition; I do not know of a formalisation of the idea as a theorem.) This holds, of course, even more so if the isomorphism is unique. For example, in $\mathcal{S}et$ all sets of the same cardinality are isomorphic, hence 'abstractly the same'. If you want to distinguish sets of the same cardinality on account of structural properties, a partial order say, you should not take $\mathcal{S}et$ as the category but another one in which the morphisms better reflect your intention. (In the case of partial orders, you could take the order-preserving functions as the morphisms, rather than all functions.)

2b Initiality and finality

All of the following definitions are relative to a category, the default one, which we don't mention explicitly to simplify the formulas. The category may and must be added to the notations, as a subscript or otherwise, in case of ambiguity.

2.6 Conventional definition. An object A is **initial** if: for each object B there is precisely one morphism from A to B, called the **mediating morphism**:

$$\forall B :: \quad \exists! f :: \quad f\colon A \to B\,.$$

Equivalently, an object A is initial if for each object B there is precisely one (at least one and at most one) solution for f in the statement

$$f\colon A \to B\,.$$

2.7 A trick. Although the formulations of the conventional definition are quite clear, they are not very well suited for algebraic manipulation. The formulation in paragraph 2.8 hasn't this drawback, as will appear from the calculations in the chapters to come. (Exercise: prove that initial objects are unique up to a unique isomorphism, and compare your proof with the one given below in paragraph 2.22.) The trick to arrive at the convenient formulation is skolemisation, named after the logician Skolem, which we'll now explain.

An assertion of the form

$$\forall\, x :: \quad \exists!\, y :: \quad \ldots y \ldots$$

is equivalent to: there exists a function $\mathcal{F}$ such that

$$(*) \qquad \forall\, x, y :: \quad \ldots y \ldots \quad \equiv \quad y = \mathcal{F}\, x\,.$$

In the former formulation it is the existential quantification ($\exists\, y$) inside the scope of a universal one that hinders effective calculation. In the latter formulation the existence claim is brought to a more global level; a reasoning need no longer be interrupted by the declaration and naming of the existence of a unique y that depends on x: it can be denoted just $\mathcal{F} x$.

In view of the important role of the various unique y's, these y's deserve a particular notation that triggers the reader of their particular properties. The notations $\mathcal{F} x$, x', and $x^\sharp$ are not specific enough. Below we employ the bracket notation $(\!|x|\!)$ and $[\![x]\!]$ for such $\mathcal{F} x$, and in the case of adjunctions we use the notation $\lfloor\!\lfloor x \rfloor\!\rfloor$ and $\lceil\!\lceil x \rceil\!\rceil$. An additional advantage of the bracket notation is that no extra parentheses are needed for composite arguments x (which we expect to occur often).

As usual we omit in line $(*)$ the universal quantifications that are outermost, thus simplifying the formulation once more.

2.8 Convenient definition. An object A is **initial** if: there exists a mapping $\llparenthesis\_\rrparenthesis$ (from objects to morphisms) such that

2.9 $\quad f\colon\ A \to B \quad \equiv \quad f = \llparenthesis B \rrparenthesis\,.$ $\qquad$ init-Charn

Mapping $\llparenthesis\_\rrparenthesis$ is called the **mediator**, and to make clear the dependency on A it is sometimes written $\llparenthesis A \to \_\rrparenthesis$. In typewriter font I would write `med( )` for $\llparenthesis\ \rrparenthesis$.

The initial object, if it exists, is unique up to a unique isomorphism (see paragraph 2.22 below); the default notation for it is 0. An alternative notation for $\llparenthesis 0 \to B \rrparenthesis$ is ¡_B.

Dually, an object A is final if, for each object B, there is precisely one morphism from B to A. In other words, an object A is **final** if: there exists a mapping $\llbracket\_\rrbracket$ such that

2.10 $\quad f\colon\ B \to A \quad \equiv \quad f = \llbracket B \rrbracket$ $\qquad$ final-Charn

Again, mapping $\llbracket\_\rrbracket$ is called the **mediator**, and it is sometimes written $\llbracket\_ \to A\rrbracket$ to make clear the dependency on A. In typewriter font I would write `dem( )`, the 'dual' of `med`.

By duality, the final object, if it exists, is unique up to a unique isomorphism; the default notation for it is 1. An alternative notation for $\llbracket B \to 1\rrbracket$ is $!_B$.

2.11 Examples. In $\mathcal{S}et$ there is just one initial object, namely the empty set. Function $\llparenthesis B \rrparenthesis$ is the 'empty function', that is, the empty set of (argument, result)-pairs. In $\mathcal{S}et$ each singleton set is a final object. Function $\llbracket B \rrbracket$ maps each $b \in B$ to the sole member of the arbitrary but fixed singleton set 1.

We shall see later that the datatype of sequences is 'the' initial object in a suitably defined category built upon $\mathcal{S}et$, and that the datatype of streams (infinite lists) is 'the' final object in another suitably defined category built upon $\mathcal{S}et$. The morphisms in these categories are homomorphisms, and the mediators $\llparenthesis\_\rrparenthesis$ and $\llbracket\_\rrbracket$ capture "definitions by induction on the structure" (structure of the argument and of the result, respectively).

We shall also see that the disjoint union and the cartesian product can be characterised by initiality and finality, respectively, in a suitably defined category built upon $\mathcal{S}et$.

2.12 Corollaries. Let A be an initial object in the category, with mediator $\llparenthesis\_\rrparenthesis$. Here are some consequences of init-Charn.

2.13 $\quad \llparenthesis A \to B \rrparenthesis\colon\ A \to B$ $\qquad$ init-Self

2.14 $\quad id_A = \llparenthesis A \to A \rrparenthesis$ $\qquad$ init-Id

2.15 $\quad f, g\colon\ A \to B \quad \Rightarrow \quad f = g$ $\qquad$ init-Uniq

2.16 $\quad f\colon\ B \to C \quad \Rightarrow \quad \llparenthesis A \to B \rrparenthesis \,;\, f = \llparenthesis A \to C \rrparenthesis$ $\qquad$ init-Fusion

Law init-Self is an instantiation of init-Charn in such a way that the right-hand side of init-Charn becomes true: take $f := \llparenthesis A \to B \rrparenthesis$. The name Self derives from the fact that $\llparenthesis B \rrparenthesis$ itSelf is a solution for x in $x\colon\ A \to B$.

Law init-Id is an instantiation of init-Charn in such a way that the left-hand side of init-Charn becomes true: take $B, f := A, id_A$.
The 'proof' of init-Uniq is left to the reader. The name Uniq derives from the fact that a solution for x in $x\colon A \to B$ is unique.
For init-Fusion we argue (suppressing A):

$$
\begin{array}{ll}
 & (\![B]\!) \mathbin{;} f = (\![C]\!) \\
\equiv & \qquad \text{init-Charn}[\, B, f := C, (\![B]\!) \mathbin{;} f \,] \\
 & (\![B]\!) \mathbin{;} f\colon\ A \to C \\
\Leftarrow & \qquad \text{composition-Type} \\
 & (\![B]\!)\colon\ A \to B \quad \wedge \quad f\colon\ B \to C \\
\equiv & \qquad \text{init-Self, and premise} \\
 & \mathit{true}.
\end{array}
$$

These five laws become much more interesting in case the category is built upon another one, $\mathcal{S}et$ for example, and the typing is expressed as one or more equations in the underlying category $\mathcal{S}et$. In particular the importance of law Fusion cannot be over-emphasised; we shall use it quite often.

Exercise: give a fully calculational proof of init-Uniq, starting with the obligation '$f = g$' at the top line of your calculation.
Exercise: give a calculational proof of the equality $(\![1]\!) = [\![0]\!]$.
Exercise: dualise the init-laws to final-laws; prove final-Fusion yourself, and see whether your proof is the dual of the one given above for init-Fusion.

2.17 Proving initiality. One may prove that an object A is initial in the category, by providing a definition for $(\![\_]\!)$ and establishing init-Charn. Almost every such a proof in the sequel has the following format. For arbitrary f and B,

$$
\begin{array}{ll}
 & f\colon\ A \to B \\
\equiv & \\
 & \vdots \\
\equiv & \\
 & f = \text{an expression not involving } f \\
\equiv & \qquad \textbf{define } (\![B]\!) = \text{ the right-hand side of the previous equation} \\
 & f = (\![B]\!).
\end{array}
$$

Actually, the last two lines in the calculation are superfluous: the remaining lines clearly show that the statement $f\colon A \to B$ has precisely one solution for f. Nevertheless, we shall not omit the last two lines for the sake of clarity. Sometimes the proof has the following format:

$$
\begin{array}{ll}
 & f\colon\ A \to B \\
\Rightarrow &
\end{array}
$$

$$
\begin{array}{ll}
 & \vdots \\
\Rightarrow & \\
 & f = \text{expression not involving } f \qquad = (\![B]\!)\ ,\ \text{ by suitably defining } (\![\_]\!) \\
\Rightarrow & \\
 & \vdots \\
\Rightarrow & \\
 & f\colon\ A \to B.
\end{array}
$$

In this case we say that we establish the *equivalence* init-Charn by *circular implication*.

In general the formulas are not as simple as suggested in the above calculations, since mostly we will be dealing with initiality in categories built upon another one, so that the typing $f\colon\ A \to B$ is a collection of equations in the underlying category.

2.18 Fact. Law init-Self says that there exists at least one morphism from A to B. Law init-Uniq says that there exists at most one morphism from A to B. Together they are equivalent to init-Charn:

2.19 $\quad$ [Self] and [Uniq] $\quad\equiv\quad$ [Charn]

where the square brackets denote the universal quantification that was implicit in the formulations above. The $\Leftarrow$-part has been argued in paragraph 2.12; for the $\Rightarrow$-part we show equivalence init-Char by circular implication:

$$
\begin{array}{lll}
 & f\colon\ A \to B & \text{(left-hand side of init-Charn)} \\
\equiv & \quad \text{init-Self} & \\
 & f\colon\ A \to B \ \text{ and } \ (\![B]\!)\colon\ A \to B & \\
\Rightarrow & \quad \text{init-Uniq} & \\
 & f = (\![B]\!) & \text{(right-hand side of init-Charn)} \\
\equiv & \quad \text{init-Self} & \\
 & f = (\![B]\!) \ \text{ and } \ (\![B]\!)\colon\ A \to B & \\
\Rightarrow & \quad \text{proposition logic, equality} & \\
 & f\colon\ A \to B & \text{(left-hand side of init-Charn)}
\end{array}
$$

In our experience, proving initiality by establishing init-Self (for some morphism denoted $(\![B]\!)$) and init-Uniq is by no means simpler or more elegant than establishing init-Charn directly, in the way explained in paragraph 2.17.

2.20 Well-formedness condition. Frequently we encounter the situation that there is a category $\mathcal{A}$ and another one, $\mathcal{B}$ say, that is built upon $\mathcal{A}$. Then the well-formedness condition for the notation $(\![B]\!)_{\mathcal{B}}$ (where B is a composite entity in the underlying category $\mathcal{A}$) is the condition that B an object in $\mathcal{B}$; this is not a purely syntactic condition.

2.21 $\quad$ B is an object in $\mathcal{B}$ $\quad\Rightarrow\quad$ $(\![B]\!)_{\mathcal{B}}$ is a morphism in $\mathcal{A}$ $\qquad$ init-Type

In the sequel we adhere to the (dangerous?) convention that in each law the free variables are quantified implicitly in such a way that the well-formedness condition, the premise of init-Type, is met.

2.22 Application. Here is an example of calculating with initiality: proving that an initial object is unique up to a unique isomorphism. Suppose that both A and B are initial. We claim that $([A \to B])$ and $([B \to A])$ establish the isomorphism and are unique in doing so. By init-Self they have the correct typing. We shall show

$$f = ([A \to B]) \ \wedge \ g = ([B \to A]) \qquad \equiv \qquad f \mathbin{;} g = id_A \ \wedge \ g \mathbin{;} f = id_B \,,$$

that is, both compositions of $([A \to B])$ and $([B \to A])$ are the identity, and conversely the identities can be factored (as in the right-hand side) only in this way. We prove both implications of the equivalence at once.

$$\begin{array}{ll}
 & f = ([A \to B]) \qquad \wedge \qquad g = ([B \to A]) \\
\equiv & \quad \text{init-Charn} \\
 & f\colon A \to B \qquad \wedge \qquad g\colon B \to A \\
\equiv & \quad \text{composition} \\
 & f \mathbin{;} g\colon A \to A \qquad \wedge \qquad g \mathbin{;} f\colon B \to B \\
\equiv & \quad \text{init-Charn} \\
 & f \mathbin{;} g = ([A \to A]) \qquad \wedge \qquad g \mathbin{;} f = ([B \to B]) \\
\equiv & \quad \text{init-Id} \\
 & f \mathbin{;} g = id_A \qquad \wedge \qquad g \mathbin{;} f = id_B.
\end{array}$$

The equality $([A \to B]) \mathbin{;} ([B \to A]) = id_A$ can be proved alternatively using init-Id, init-Fusion, and init-Self in that order. (This gives a nice proof of the weaker claim that initial objects are isomorphic.)

2c Products and Sums

Products and sums are dual categorical concepts that, specialised to category $\mathcal{S}et$, yield the well-known notions of cartesian product and disjoint union. (In other categories products and sums may get a different interpretation.)

2.23 Disjoint union. As an introduction to the definition of the categorical sum, we present here a categorical description of the disjoint union. Let the default category be Set. The disjoint union of sets A and B is a set, usually called $A + B$, with several operations associated with it. There are the injections

$$\begin{array}{l}
inl\colon A \to A + B \\
inr\colon B \to A + B\,,
\end{array}$$

and there may be a predicate that tests whether an element in $A + B$ is $inl(x)$ or $inr(y)$ for some $x \in A$ or some $y \in B$. Using the predicate one can define an operation that in programming languages is known as a **case** construct, and vice versa. The case construct of f and g is denoted $f \triangledown g$ and has the following typing and semantics.

$$f \triangledown g\colon\ A + B \to C \qquad\qquad \text{for } f\colon\ A \to C \ \text{ and } \ g\colon\ B \to C$$

and

$$\begin{array}{lcll} \mathit{inl} \,;\, f \triangledown g & = & x \mapsto f\,x & \text{for each } x \in A \\ \mathit{inr} \,;\, f \triangledown g & = & y \mapsto g\,y & \text{for each } y \in B\,. \end{array}$$

By extensionality the two equations read:

$$\begin{array}{lcl} \mathit{inl} \,;\, f \triangledown g & = & f \\ \mathit{inr} \,;\, f \triangledown g & = & g\,. \end{array}$$

Moreover, $f \triangledown g$ is the only solution for h in the two equations:

$$\begin{array}{lcl} \mathit{inl} \,;\, h & = & f \\ \mathit{inr} \,;\, h & = & g\,. \end{array}$$

This is an important observation; it holds for each representation of disjoint unions! Indeed, a 'disjoint union'-like concept for which the claim does not hold, is normally not considered to be a proper 'disjoint union' of A and B.
Exercise: consider the representation $A + B = \{0\} \times A \cup \{1\} \times B$, and work out operations inl, inr, and $f \triangledown g$. Also, think of another representation for $A + B$, and work out the operations again. Prove in each case the above claims. Would you call $A \cup B$ a disjoint union of A and B? Why, or why not?

In summary, we call functions $\mathit{inl}\colon\ A \to D$ and $\mathit{inr}\colon\ B \to D$ together with their target D a disjoint union of A and B if, and only if, for each $f\colon\ A \to C$ and $g\colon\ B \to C$ there is precisely one function h, henceforth denoted $f \triangledown g$, such that

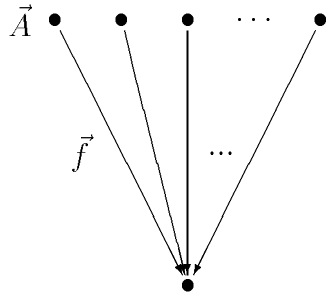
2.24 $\qquad \mathit{inl} \,;\, h = f \ \wedge\ \mathit{inr} \,;\, h = g\,.$

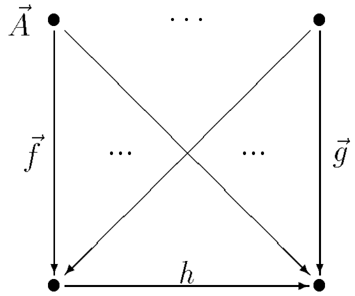
This is an entirely categorical formulation. In addition, the formulation suggests to look for a characterisation by means of initiality (or finality). With a suitable definition for $\bigvee(A, B)$ (given below), the above pair of equations can be formulated as

2.25 $\qquad h\colon\ (\mathit{inl}, \mathit{inr}) \to_{\bigvee(A,B)} (f, g)\,.$

So $(\mathit{inl}, \mathit{inr})$ is initial in $\bigvee(A, B)$. Having available the pair $(\mathit{inl}, \mathit{inr})$ (as 'the' initial object in $\bigvee(A, B)$), the set $A + B$ can be defined by $A + B = \mathrm{tgt}\ \mathit{inl} = \mathrm{tgt}\ \mathit{inr}$. Thus the notion of disjoint union has been characterised categorically, by initiality, and it turns out that the injections $\mathit{inl}, \mathit{inr}$ and operation $\triangledown$ are as relevant for the notion of disjoint union of A and B as the set $A + B$ itself.

2.26 Category $\bigvee(\vec{A})$. Let $\mathcal{A}$ be a category, the default one; in the above discussion we had $\mathcal{A} = \mathcal{S}et$. Let $\vec{A}$ be an n-tuple of objects. Then category $\bigvee(\vec{A})$ is: the category built upon $\mathcal{A}$ with the following objects, morphisms, and typing. An object in $\bigvee(\vec{A})$ is: an n-tuple of morphisms in $\mathcal{A}$ with a common target and the objects $\vec{A}$ as sources.





Let $\vec{f}$ and $\vec{g}$ be such objects; then a morphism from $\vec{f}$ to $\vec{g}$ in $\bigvee(\vec{A})$ is: a morphism h in $\mathcal{A}$ satisfying $f_i \,;\, h = g_i$ for each index i of the n-tuple. It follows that h: tgt $\vec{f} \to_{\mathcal{A}}$ tgt $\vec{g}$.

Exercise: spell out the definition of $\bigvee(A)$. This category is commonly called the 'co-slice' category 'under A'.
Exercise: does the text above define a category or a pre-category?
Exercise: spell out the definition of $\bigvee(A, B)$, taking $\mathcal{A}$ to be $\mathcal{S}et$. Verify the equivalence of formulas 2.24 and 2.25.
Exercise: check the sensefulness of the following definition. A **parallel pair** with source A is: an object in $\bigvee(A, A)$.
Exercise: define $\bigwedge(\vec{A})$ dually to $\bigvee(\vec{A})$.

Having discussed a categorical characterisation (definition) of disjoint unions, we now abstract from $\mathcal{S}et$, and thus obtain a definition of sums.

2.27 Sum. Let $\mathcal{A}$ be an arbitrary category, the default category, and let A, B be objects. A **sum** of A and B is: an initial object in $\bigvee(A, B)$; it may or may not exist. Let (inl, inr) be a sum of A and B; their common target is denoted $A + B$. We abbreviate $(\![(inl, inr) \to (f, g)]\!)_{\bigvee(A,B)}$ to just $f \triangledown g$, not mentioning the dependency on A, B and inl, inr. Working out 'being an object in $\bigvee(A, B)$' in terms of $\mathcal{A}$ yields the following instantiation of init-Type:

$$f\colon A \to C \ \wedge\ g\colon B \to C \quad \Rightarrow \quad f \triangledown g\colon A + B \to C \qquad \triangledown\text{-Type}$$

Working out the typing in $\bigvee(A, B)$ in terms of equations in $\mathcal{A}$ yields the following instantiations of the laws for initiality:

$$inl \,;\, f = g \ \wedge\ inr \,;\, f = h \qquad \equiv \qquad f = g \triangledown h \qquad \triangledown\text{-Charn}$$

$$inl \,;\, f \triangledown g = f \ \wedge\ inr \,;\, f \triangledown g = g \qquad \triangledown\text{-Self}$$

$$inl \triangledown inr = id \qquad \triangledown\text{-Id}$$

$$inl \,;\, f = inl \,;\, g \;\wedge\; inr \,;\, f = inr \,;\, g \qquad \Rightarrow \qquad f = g \qquad \triangledown\text{-Uniq}$$

$$f \,;\, k = h \;\wedge\; g \,;\, k = j \qquad \Rightarrow \qquad f \triangledown g \,;\, k = h \triangledown j \qquad \triangledown\text{-Fusion}$$

Law $\triangledown$-Uniq says that the pair inl, inr is *jointly epic*. Law $\triangledown$-Fusion simplifies to an unconditional law by substituting $h, j := f \,;\, h,\ g \,;\, h$:

$$f \triangledown g \,;\, h = (f \,;\, h) \triangledown (g \,;\, h) \qquad \triangledown\text{-Fusion}$$

Similar simplifications will be done tacitly in the sequel.

Notice that for given $f\colon A + B \to C$ the equation $x \triangledown y = f$ *defines* x and y, since by $\triangledown$-Charn that one equation equivales the two equations $x = inl \,;\, f$ and $y = inr \,;\, f$. We shall quite often use this form of definition.

The usual categorical notation for $f \triangledown g$ is $[f, g]$; the symbol $\triangledown$ was first used for this purpose by Fokkinga and Meijer [5]. Operation $\triangledown$ is sometimes called *junc*, from junction; I myself like the name **dis**, from disjunction, and in typewriter font I would write `dis`. Morphisms inl and inr are called **injections**. In the case of the straightforward generalisation of an n-fold sum, we denote the injections by $in_0, \ldots, in_{n-1}$, possibly decorated with n as well.

Exercise: verify that all five $\triangledown$-laws above are instantiations of the laws for initiality by substituting, amongst others, $\mathcal{A}, A := \bigvee(A, B), (inl, inr)$ in the init-laws.
Exercise: take $\mathcal{A} = \mathcal{S}et$, so that a sum of two sets is a disjoint union of the sets, and prove the laws Self, ... Fusion in set-theoretic terms for one particular representation for the disjoint union.
Exercise: initial objects are unique up to a unique isomorphism; work out in terms of $\mathcal{A}$ what that means for (inl, inr).

2.28 Product. Products are, by definition, dual to sums. Let exl, exr be a product of A and B, supposing one exists; its common source is denoted $A \times B$. We abbreviate $[\![f, g \leftarrow exl, exr]\!]_{\bigwedge(A,B)}$ to just $f \vartriangle g$. The typing law works out as follows:

$$f\colon C \to A \;\wedge\; g\colon C \to B \quad \Rightarrow \quad f \vartriangle g\colon C \to A \times B \qquad \vartriangle\text{-Type}$$

The laws for exl, exr and $\vartriangle$ work out as follows:

$$f \,;\, exl = g \;\wedge\; f \,;\, exr = h \qquad \equiv \qquad f = g \vartriangle h \qquad \vartriangle\text{-Charn}$$

$$f \vartriangle g \,;\, exl = f \;\wedge\; f \vartriangle g \,;\, exr = g \qquad \vartriangle\text{-Self}$$

$$exl \vartriangle exr = id \qquad \vartriangle\text{-Id}$$

$$f \,;\, exl = g \,;\, exl \;\wedge\; f \,;\, exr = g \,;\, exr \qquad \Rightarrow \qquad f = g \qquad \vartriangle\text{-Uniq}$$

$$f \,;\, g \vartriangle h = (f \,;\, g) \vartriangle (f \,;\, h) \qquad \vartriangle\text{-Fusion}$$

Law $\vartriangle$-Uniq says that the pair (exl, exr) is *jointly monic*. Law $\vartriangle$-Fusion has been simplified to an unconditional form.

Notice that for given $f\colon A \to B \times C$ the equation $x \vartriangle y = f$ *defines* x and y, since by $\vartriangle$-Charn that one equation equivales the two equations $x = f \mathbin{;} \mathit{exl}$ and $y = f \mathbin{;} \mathit{exr}$.

The usual categorical notation for $f \vartriangle g$ is $\langle f, g\rangle$; the symbol $\vartriangle$ was first used for this purpose by Fokkinga and Meijer [5]. Operation $\vartriangle$ is sometimes called *split*; I myself like the name **con**, from conjunction, and in typewriter font I would write `con`. Morphisms exl and exr are called **extractions**. In the case of the straightforward generalisation of an n-fold product, we denote the extractions by $\mathit{ex}_0, \ldots, \mathit{ex}_{n-1}$, possibly decorated with n as well.

Exercise: check that these laws are the dual of those for sums.

2.29 Application. As an application of the laws for sum and product we show that $\triangledown$ and $\vartriangle$ abide. Two binary operations φ and $\ominus$ **abide** with each other if: for all values a, b, c, d

$$(a \varphi b) \ominus (c \varphi d) \quad = \quad (a \ominus c) \varphi (b \ominus d)\,.$$

Writing $a \varphi b$ as $a \mid b$ and $a \ominus b$ as $\frac{a}{b}$, the equation reads

$$\frac{a \mid b}{c \mid d} \quad = \quad \frac{a}{c} \Big| \frac{b}{d}\,.$$

The term *abide* has been coined by Bird [3] and comes from "above-beside." In category theory this property is called a 'middle exchange rule' or 'interchange rule'.

Here is a proof that $\triangledown$ and $\vartriangle$ abide:

$$\begin{array}{cl}
 & (f \triangledown g) \vartriangle (h \triangledown j) = (f \vartriangle h) \triangledown (g \vartriangle j) \\
\equiv & \qquad \triangledown\text{-Charn } [f, g, h := \text{lhs},\ f \vartriangle h,\ g \vartriangle j] \\
 & \mathit{inl} \mathbin{;} (f \triangledown g) \vartriangle (h \triangledown j) = f \vartriangle h \quad \wedge \quad \mathit{inr} \mathbin{;} (f \triangledown g) \vartriangle (h \triangledown j) = g \vartriangle j \\
\equiv & \qquad \vartriangle\text{-Fusion (at two places)} \\
 & (\mathit{inl} \mathbin{;} f \triangledown g) \vartriangle (\mathit{inl} \mathbin{;} h \triangledown j) = f \vartriangle h \quad \wedge \quad (\mathit{inr} \mathbin{;} f \triangledown g) \vartriangle (\mathit{inr} \mathbin{;} h \triangledown j) = g \vartriangle j \\
\equiv & \qquad \triangledown\text{-Self (at four places)} \\
 & f \vartriangle h = f \vartriangle h \quad \wedge \quad g \vartriangle j = g \vartriangle j \\
\equiv & \qquad \text{equality} \\
 & \mathit{true}.
\end{array}$$

Exercise: give another proof in which you start with $\vartriangle$-Charn rather than $\triangledown$-Charn.
Exercise: give another proof in which you start as above and then apply $\vartriangle$-Charn at the second step (at two places).
Exercise: choose an explicit representation for the disjoint union (and cartesian product), and prove the abides law in set-theoretic terms, using the chosen representation.

2.30 More laws. For arbitrary categories in which sums and products, respectively, exist, we define, for $f\colon A \to B$ and $g\colon C \to D$,

$$\begin{array}{lclcl} f + g &=& (f \,;\, \mathit{inl}) \triangledown (g \,;\, \mathit{inr}) &:& A + C \to B + D \\ f \times g &=& (\mathit{exl} \,;\, f) \vartriangle (\mathit{exr} \,;\, g) &:& A \times C \to B \times D\,. \end{array}$$

In case the category is $\mathcal{S}et$, function $f \times g$ acts componentwise: $(a,b) \overset{f\times g}{\mapsto} (fa, gb)$; similarly, $\mathit{inl}(a) \overset{f+g}{\mapsto} \mathit{inl}(fa)$ and $\mathit{inr}(b) \overset{f+g}{\mapsto} \mathit{inr}(gb)$. The mappings $+$ and $\times$ are bifunctors: $id + id = id$ and $f + g \;;\; h + j = (f \,;\, h) + (g \,;\, j)$, and similarly for $\times$. Throughout the text we shall use several properties of product and sum. These are referred to by the hint 'product' or 'sum'. Here is a list; some of these are just the laws presented before.

$$\begin{array}{rcl@{\qquad}rcl} f \times g \,;\, \mathit{exl} &=& \mathit{exl} \,;\, f & \mathit{inl} \,;\, f + g &=& f \,;\, \mathit{inl} \\ f \vartriangle g \,;\, \mathit{exl} &=& f & \mathit{inl} \,;\, f \triangledown g &=& f \\ f \times g \,;\, \mathit{exr} &=& \mathit{exr} \,;\, g & \mathit{inr} \,;\, f + g &=& g \,;\, \mathit{inr} \\ f \vartriangle g \,;\, \mathit{exr} &=& g & \mathit{inr} \,;\, f \triangledown g &=& g \\ f \,;\, g \vartriangle h &=& (f \,;\, g) \vartriangle (f \,;\, h) & f \triangledown g \,;\, h &=& (f \,;\, h) \triangledown (g \,;\, h) \\ \mathit{exl} \vartriangle \mathit{exr} &=& id & \mathit{inl} \triangledown \mathit{inr} &=& id \\ (h \,;\, \mathit{exl}) \vartriangle (h \,;\, \mathit{exr}) &=& h & (\mathit{inl} \,;\, h) \triangledown (\mathit{inr} \,;\, h) &=& h \\ f \vartriangle g \,;\, h \times j &=& (f \,;\, h) \vartriangle (g \,;\, j) & f + g \,;\, h \triangledown j &=& (f \,;\, h) \triangledown (g \,;\, j) \\ f \times g \,;\, h \times j &=& (f \,;\, h) \times (g \,;\, j) & f + g \,;\, h + j &=& (f \,;\, h) + (g \,;\, j) \\ f \vartriangle g = h \vartriangle j &\equiv& f = h \wedge g = j & f \triangledown g = h \triangledown j &\equiv& f = h \wedge g = j \end{array}$$

Exercise: identify the laws that we've seen already, and prove the others.
Exercise: above we've explained $f \times g$ and $f + g$ in set-theoretic terms in case the category is $\mathcal{S}et$; which of the equations comes closest to those specifications "at the point-level"?
Exercise: what about the following equivalences:

$$f \times g = h \times j \;\overset{?}{\equiv}\; f = h \wedge g = j \qquad f + g = h + j \;\overset{?}{\equiv}\; f = h \wedge g = j$$

Are these true in each category? (Answer: no. Hint: in $\mathcal{S}et$ we have $A \times \emptyset = \emptyset$ for each A. Now take $f, h\colon A \to A$ arbitrary, and $g = j = id_\emptyset$.)
Exercise: prove that in each category each $\mathit{exl}_{A,A}$ is epic, whereas $\mathit{exl}_{A,B}$ is not necessarily epic. (Hint: take $B = \emptyset$ in $\mathcal{S}et$.)
Exercise: prove that in each category that has products and a final object, $1 \times A \cong A$ and $A \times (B \times C) \cong (A \times B) \times C$.

2d Coequalisers

As another example of a categorical characterisation by initiality, we present here the notion of coequaliser. A coequaliser is a categorical notion that, specialised to category $\mathcal{S}et$, yields the well-known and important notion of induced equivalence relation.

2.31 Category $\bigvee(f \parallel g)$. In order to characterise coequalisers by initiality, we need the auxiliary notion of $\bigvee(f \parallel g)$.

Let $\mathcal{A}$ be a category, the default one (think for example of $\mathcal{S}et$). Let (f, g) be a parallel pair, that is, f and g have a common source and a common target. Then $\bigvee(f \parallel g)$ is the category built upon $\mathcal{A}$ as follows. An object in $\bigvee(f \parallel g)$ is: a morphism p in $\mathcal{A}$ satisfying $f ; p = g ; p$. Let p and q be objects in $\bigvee(f \parallel g)$; then a morphism in $\bigvee(f \parallel g)$ from p to q is: a morphism x in $\mathcal{A}$ such that $p ; x = q$.

$$\begin{tikzcd}
\bullet \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & \bullet \arrow[r, "p"] \arrow[dr, "q"'] & \bullet \arrow[d, "x"] \\
& & \bullet
\end{tikzcd}$$

The phrase 'p is an object in $\bigvee(f \parallel g)$' is just a concise way of saying 'p is a morphism satisfying $f ; p = g ; p$'. Unfortunately there is no simple noun or verb for this property.

2.32 Definition. Let $\mathcal{A}$ be a category, the default one. Let (f, g) be a parallel pair. Then, a **coequaliser** of f, g is: an initial object in $\bigvee(f \parallel g)$.

Let p be a coequaliser of (f, g), supposing one exists. We write $p\backslash_{f,g}q$ or simply $p\backslash q$ instead of $\llparenthesis p \to q \rrparenthesis_{\bigvee(f\|g)}$ since, as we shall explain, the fraction notation better suggests the calculational properties. Working out the definition of being an object in $\bigvee(f \parallel g)$ in terms of equations in $\mathcal{A}$, we obtain the following instantiation of the laws for initiality.

$$f ; q = g ; q \quad \Rightarrow \quad p\backslash q\colon\ \text{tgt}\, p \to \text{tgt}\, q \qquad \backslash\text{-Type}$$

and further:

$$p ; x = q \qquad \equiv \qquad x = p\backslash q \qquad \backslash\text{-Charn}$$

$$p ; p\backslash q = q \qquad \backslash\text{-Self}$$

$$id = p\backslash p \qquad \backslash\text{-Id}$$

$$p ; x = q \ \wedge\ p ; y = q \qquad \Rightarrow \qquad x = y \qquad \backslash\text{-Uniq}$$

$$q ; x = r \qquad \Rightarrow \qquad p\backslash q ; x = p\backslash r \qquad \backslash\text{-Fusion}$$

In accordance with the convention explained in paragraph 2.20 we have omitted in laws $\backslash$-Charn, $\backslash$-Self and $\backslash$-Fusion the well-formedness condition that q is an object in $\bigvee(f\|g)$; the notation $...\backslash q$ is only senseful if $f ; q = g ; q$, like in arithmetic where the notation m/n is only senseful if n differs from 0. Notice that $\backslash$-Uniq and $\backslash$-Fusion simplify to:

$$p ; x = p ; y \qquad \Rightarrow \qquad x = y \qquad \backslash\text{-Uniq}$$

$$p\backslash q ; x = p\backslash(q ; x) \qquad \backslash\text{-Fusion}$$

Thus, $\backslash$-Uniq expresses that a coequaliser is epic.

2.33 Additional laws. The following law confirms the choice of notation once more.

$$p\backslash q \mathbin{;} q\backslash r = p\backslash r \qquad \backslash\text{-Compose}$$

Here is one way to prove it.

$$\begin{array}{ll} & p\backslash q \mathbin{;} q\backslash r \\ = & \qquad \backslash\text{-Fusion} \\ & p\backslash(q \mathbin{;} q\backslash r) \\ = & \qquad \backslash\text{-Self} \\ & p\backslash r. \end{array}$$

An interesting aspect is that the omitted subscripts to $\backslash$ may differ: e.g., $p\backslash_{f,g}q$ and $q\backslash_{h,j}r$, and q is not necessarily a coequaliser of f,g. Rephrased in the notation for initiality in general, law $\backslash$-Compose reads:

2.34 $$(\!|A \to B|\!)_{\mathcal{A}} \mathbin{;} (\!|B \to C|\!)_{\mathcal{B}} = (\!|A \to C|\!)_{\mathcal{A}} \qquad \text{init-Compose}$$

where $\mathcal{A}$ and $\mathcal{B}$ are full subcategories of some category $\mathcal{C}$ and objects B, C are in both $\mathcal{A}$ and $\mathcal{B}$; in our case $\mathcal{A} = \bigvee(f \| g)$, $\mathcal{B} = \bigvee(h \| j)$, and $\mathcal{C} = \bigvee(D)$ where D is the common target of f, g, h, j. Then the proof runs as follows.

$$\begin{array}{ll} & (\!|A \to B|\!)_{\mathcal{A}} \mathbin{;} (\!|B \to C|\!)_{\mathcal{B}} = (\!|A \to C|\!)_{\mathcal{A}} \\ \Leftarrow & \qquad \text{init-Fusion} \\ & (\!|B \to C|\!)_{\mathcal{B}} \colon B \to_{\mathcal{A}} C \\ \equiv & \qquad \text{both } \mathcal{A} \text{ and } \mathcal{B} \text{ are full subcategories of } \mathcal{C}, \\ & \qquad \text{each containing both } B \text{ and } C \\ & (\!|B \to C|\!)_{\mathcal{B}} \colon B \to_{\mathcal{B}} C \\ \equiv & \qquad \text{init-Self} \\ & \mathit{true}. \end{array}$$

Here is another law; its proof shows two of the above laws in action. As before, let p be a coequaliser. Then

$$F(p\backslash q) = Fp\backslash Fq \qquad \backslash\text{-Ftr}$$

The implicit well-formedness condition here is that Fp is a coequaliser. Clearly, this condition is valid when F preserves coequalisers. The proof of the law reads:

$$\begin{array}{ll} & F(p\backslash q) = Fp\backslash Fq \\ \equiv & \qquad \backslash\text{-Charn} \\ & Fp \mathbin{;} F(p\backslash q) = Fq \\ \equiv & \qquad \text{functor} \end{array}$$

$$\begin{array}{ll} & F(p \mathbin{;} p\backslash q) = Fq \\ \equiv & \qquad \backslash\text{-Self} \\ & \mathit{true}. \end{array}$$

Exercise: let p be a coequaliser of a parallel pair (f, g), and let h be an epimorphism with $\operatorname{tgt} h = \operatorname{src} f = \operatorname{src} g$. Prove that p is a coequaliser of $(h \mathbin{;} f,\ h \mathbin{;} g)$.
Exercise: let p_i be a coequaliser of a pair (f_i, g_i), for $i = 0, 1$. Prove that $p_0 + p_1$ is a coequaliser of $(f_0 + f_1,\ g_0 + g_1)$, assuming that sums exist.

2.35 Interpretation in $\mathcal{S}et$. Take $\mathcal{A} = \mathcal{S}et$, the default category, and let A be a set. Each parallel pair (f, g) with target A determines a binary relation $R_{f,g}$ on A, and conversely, each binary relation R on A determines a parallel pair (f_R, g_R) in the following way:

$$\begin{array}{lcll} R_{f,g} & = & \{(fx, gx) \mid x \in \operatorname{src} f\} & \subseteq A \times A \\ f_R & = & \mathit{exl} & : \{(x, y) \mid x\, R\, y\} \to A \\ g_R & = & \mathit{exr} & : \{(x, y) \mid x\, R\, y\} \to A\,. \end{array}$$

In this way parallel pairs with target A represent binary relations on A.

In a similar but different way, functions with source A (that is, objects in $\bigvee(A)$) represent equivalence relations. Specifically, each function q with source A determines an equivalence relation E_q on A, and conversely, each equivalence relation E on A determines such a function q_E, in the following way:

$$\begin{array}{lcll} E_q & = & \{(x, y) \mid qx = qy\} & \subseteq A \times A \\ q_E & = & x \mapsto \text{the } E\text{-equivalence class containing } x & : A \to A/E\,. \end{array}$$

Exercise: check, or prove, each of the following claims. Category $\bigvee(f \parallel g)$ is a subcategory of $\bigvee(A)$. For object p in $\bigvee(A)$, relation E_p is an equivalence relation. For objects p, q in $\bigvee(A)$, there exists a function x with $x\colon p \to_{\bigvee(A)} q$ if and only if $E_p \subseteq E_q$. For each object q in $\bigvee(f \parallel g)$, $R_{f,g} \subseteq E_q$. Let p be a coequaliser of (f, g); then E_p is the least equivalence relation including $R_{f,g}$, that is, E_p is the equivalence relation *induced* by $R_{f,g}$.

Exercise: let p be a coequaliser of a parallel pair (f, g) with target A, and define B and $q\colon A \to B$ by:

$$\begin{array}{lcl} B & = & \operatorname{tgt} p \cup \{\bullet\} \\ q & = & x \mapsto px, \quad \text{for each } x \in A \\ & : & A \to B\,. \end{array}$$

Is q also a coequaliser of (f, g)? (Hint: by construction, q is not surjective.) Give x and y satisfying $x\colon p \to_{\bigvee(A)} q$ and $y\colon q \to_{\bigvee(A)} p$, if they exist. Let r be an object in $\bigvee(f \parallel g)$; prove or disprove the existence of at least one solution for x in $x\colon q \to_{\bigvee(f \parallel g)} r$, and prove or disprove the existence of at most one solution for x.

2e Colimits

The notion of colimit is a far-reaching generalisation of the notions of sum, coequaliser, and several others. Each colimit is a certain initial object, and each initial object is a certain colimit. We shall briefly define the notion of colimit, and present its calculational properties derived from the characterisation by initiality. We shall also give a nontrivial application involving colimits.

By definition, limits are dual to colimits. So by duality limits generalise such notions as product, equaliser, and several others. Each limit is a certain final object, and each final object is a certain limit.

The formal definition uses the notions of a diagram D and of the cocone category $\bigvee D$, which we now present.

2.36 Diagram. Let $\mathcal{A}$ be a category, the default one. A **diagram** in $\mathcal{A}$ is: a graph whose nodes are labelled with objects and whose edges are labeled with morphisms, in such a way that the labeling is "consistent" with the typing of the category, that is, for labeling $\overset{A}{\bullet} \xrightarrow{f} \overset{B}{\bullet}$ in the diagram, it is required that $f\colon A \to B$ in the category. As a consequence, if $\xrightarrow{f} \bullet \xrightarrow{g}$ is in the diagram, then $\operatorname{tgt} f = \operatorname{src} g$ in the category.

Although a diagram in $\mathcal{A}$ is (or: determines) a category, that category is not necessarily a subcategory of $\mathcal{A}$; distinct edges may have the same label. Here is a counterexample. Let $\mathcal{A}$ be the category determined by:

$$
\begin{tikzcd}
\overset{A}{\bullet} \arrow[r, "f"] & \overset{B}{\bullet} \arrow[r, "g"] & \overset{C}{\bullet}
\end{tikzcd}
$$

and put $h = f \mathbin{;} g$. Then these are diagrams in $\mathcal{A}$:

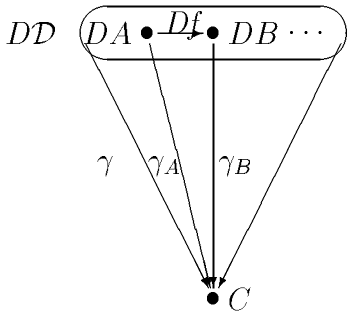
$$
\begin{tikzcd}
\overset{A}{\bullet} \arrow[r, shift left, "f"] \arrow[r, shift right, "f"'] & \overset{B}{\bullet}
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\overset{A}{\bullet} \arrow[r, "f"] \arrow[rr, bend right, "h"'] & \overset{B}{\bullet} \arrow[r, "g"] & \overset{C}{\bullet}
\end{tikzcd}
$$

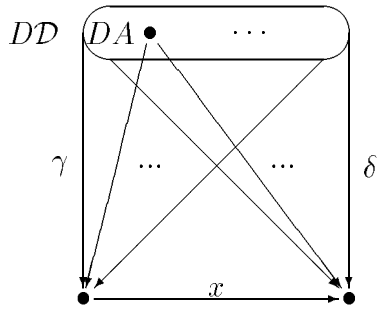
In the left diagram there are two edges (morphisms) from A to B, whereas in $\mathcal{A}$ there is only one. In the right diagram there are two edges (morphisms) from A to C, labelled $f \mathbin{;} g$ and h respectively, whereas in $\mathcal{A}$ there is only one; by definition $h = f \mathbin{;} g$.

Extreme cases of diagrams are diagrams with zero, one, or more nodes only, and no edges at all.

For simplicity in the formulations to come, we consider a diagram in $\mathcal{A}$ to be a functor $D\colon \mathcal{D} \to \mathcal{A}$, where $\mathcal{D}$ is a graph (hence category) giving the shape of the diagram in $\mathcal{A}$, and D gives the labeling: a node A in $\mathcal{D}$ is labeled DA (an object in $\mathcal{A}$), and an edge f in $\mathcal{D}$ is labeled Df (a morphism in $\mathcal{A}$).

2.37 Category $\bigvee D$. Let $\mathcal{A}$ be a category, the default one, and let $D\colon \mathcal{D} \to \mathcal{A}$ be a functor, hence diagram in $\mathcal{A}$. Then category $\bigvee D$, built upon $\mathcal{A}$, is defined as follows; its objects are called **cocones** for D.





A cocone for D is: a family $\gamma_A\colon DA \to C$ of morphisms (for some C), one for each A in $\mathcal{D}$, satisfying:

$$Df \mathbin{;} \gamma_B \quad = \quad \gamma_A \qquad \text{for each } f\colon A \to_{\mathcal{D}} B\,.$$

This condition is called 'commutativity of the triangles'. Using naturality and constant functors there is a technically simpler definition of a cocone. Define $\underline{C}$ to be the constant functor, $\underline{C}\,x = C$ for each object x, and $\underline{C}\,f = id_C$ for each morphism f. Now, each cocone for D is a natural transformation $\gamma\colon D \mathrel{\dot\to} \underline{C}$ in $\mathcal{A}$ (for some C), and vice versa. (Exercise: check this.) We define, for the γ above, $\operatorname{tgt}\gamma = C$. (Notice that γ is a morphism in the functor category $\mathcal{F} = \mathcal{F}tr(\mathcal{D}, \mathcal{A})$, so that $\operatorname{tgt}_{\mathcal{F}}\gamma = \underline{C}$. The object C really forms part of the cocone, even if $\mathcal{D}$ is empty and, hence, γ is an empty family. To stress this fact, one might prefer to define a cocone as a pair (C, γ).)

Continuing the definition of category $\bigvee D$, let γ and δ be cocones for D; then, a morphism from γ to δ in $\bigvee D$ is: a morphism x satisfying $\gamma_A \mathbin{;} x = \delta_A$ for each object A in $\mathcal{D}$. It follows that $x\colon \operatorname{tgt}\gamma \to \operatorname{tgt}\delta$. The condition on x can be written simply $\gamma \mathbin{;} x = \delta$, when we define the composition of a cocone with a morphism by:

$$(\gamma \mathbin{;} x)_A \quad = \quad \gamma_A \mathbin{;} x\,. \qquad\qquad \text{(composition of a cocone with a morphism)}$$

Then

$$\gamma\colon D \mathrel{\dot\to} \underline{C} \ \text{ and } \ x\colon C \to C' \qquad \Rightarrow \qquad \gamma \mathbin{;} x\colon D \mathrel{\dot\to} \underline{C'}\,.$$

(An alternative would be to write $\gamma \mathbin{;} \underline{x}$, since for $x\colon C \to C'$ the constant function $\underline{x} = A \mapsto x$ is a natural transformation of type $\underline{C} \mathrel{\dot\to} \underline{C'}$.)

2.38 Definition. Let $\mathcal{A}$ be a category, the default one, and let D diagram in $\mathcal{A}$. A **colimit** for D is: an initial object in $\bigvee D$; it may or may not exist.

Let γ be a colimit for D. We write $(\![\_]\!)$ as $\gamma\backslash\_$. Working out the definition of cocone in terms of equations in $\mathcal{A}$, we obtain the following characterisation of a colimit. There exists a mapping $\gamma\backslash\_$ such that

$$\delta \text{ cocone for } D \qquad \Rightarrow \qquad \gamma\backslash\delta\colon \operatorname{tgt}\gamma \to \operatorname{tgt}\delta \qquad\qquad \backslash\text{-Type}$$

and further $\backslash$-Charn holds, and its corollaries too:

$$\gamma \,;\, x = \delta \qquad \equiv \qquad x = \gamma\backslash\delta \qquad\qquad \backslash\text{-Charn}$$
$$\gamma \,;\, \gamma\backslash\delta = \delta \qquad\qquad \backslash\text{-Self}$$
$$\gamma\backslash\gamma = id \qquad\qquad \backslash\text{-Id}$$
$$\gamma \,;\, x = \gamma \,;\, y \qquad \Rightarrow \qquad x = y \qquad\qquad \backslash\text{-Uniq}$$
$$\gamma\backslash\delta \,;\, x = \gamma\backslash(\delta \,;\, x) \qquad\qquad \backslash\text{-Fusion}$$
$$\gamma\backslash\delta \,;\, \delta\backslash\varepsilon = \gamma\backslash\varepsilon \qquad\qquad \backslash\text{-Compose}$$
$$F(\gamma\backslash\delta) = F\gamma\backslash F\delta \qquad\qquad \backslash\text{-Ftr}$$

for D-cocones δ and ε (δ and $F\gamma$ being a colimit when occurring as the left argument of $\backslash$.) Law $\backslash$-Uniq asserts that each colimit is 'jointly epic'. For the proof of $\backslash$-Compose and $\backslash$-Ftr see law init-Compose and $\backslash$-Ftr in paragraph 2.33.

2.39 Another law. Here is another law. Write the subscripts to natural transformations as proper arguments: $\gamma_A = \gamma A$, and recall the definition $(\gamma F)A = \gamma(FA)$, for a functor F. Let both γ and γF be colimits (for the same diagram). Then, for each cocone δ for that same diagram:

$$\gamma F\backslash\delta F \quad = \quad \gamma\backslash\delta$$

Here is the proof:

$$\begin{array}{ll}
 & \gamma F\backslash\delta F = \gamma\backslash\delta \\
\equiv & \qquad \backslash\text{-Charn } [\gamma, \delta, x := \gamma F, \delta F, \gamma\backslash\delta] \\
 & \gamma F \,;\, \gamma\backslash\delta = \delta F \\
\equiv & \qquad \backslash\text{-Self applied within the right-hand side: } \delta = \gamma \,;\, \gamma\backslash\delta \\
 & \gamma F \,;\, \gamma\backslash\delta = (\gamma \,;\, \gamma\backslash\delta)F \\
\equiv & \qquad \text{extensionality} \\
 & (\gamma F \,;\, \gamma\backslash\delta)A = (\gamma \,;\, \gamma\backslash\delta)FA \quad \text{for each } A \\
\equiv & \qquad \text{composition of cocone with a morphism} \\
 & \mathit{true}.
\end{array}$$

We shall now show in paragraphs 2.40, 2.41, and 2.42 that initial objects, sums, and coequalisers are colimits. Then we give in paragraph 2.43 an example of a colimit that explains the term 'limit'. Finally in paragraph 2.44 we give an nontrivial application of the laws for colimits. In paragraph A.55 it is shown that left adjoints preserve colimits.

2.40 Initiality as colimit. Let $\mathcal{A}$ be a category, the default one. Take $\mathcal{D}$ empty, so that $D\colon \mathcal{D} \to \mathcal{A}$ is the empty functor. Then a cocone δ for D is the empty family $(\,)_B$ of morphisms, where $B = \mathrm{tgt}\,\delta$.

Suppose $\gamma = (\,)_A$ is a colimit for D; it may or may not exist. Then A is initial in $\mathcal{A}$. To show this, we establish init-Charn, constructing $(\!|\_|\!)$ along the way. For arbitrary object B and morphism x,

$$
\begin{array}{ll}
 & x\colon A \to B \\
\equiv & \quad \text{property of the empty natural transformations } (\,)_A \text{ and } (\,)_B \\
 & (\,)_A \,;\, x = (\,)_B \\
\equiv & \quad \gamma = (\,)_A \text{ is colimit for } D\text{, and } (\,)_B \text{ is cocone for } D\text{; } \backslash\text{-Charn} \\
 & x = \gamma\backslash(\,)_B \\
\equiv & \quad \textbf{define } (\!|B|\!) = \gamma\backslash(\,)_B \\
 & x = (\!|B|\!).
\end{array}
$$

Exercise: show that, if there exists an initial object in $\mathcal{A}$, then there exists a colimit for the diagram D above.

2.41 Sum as colimit. Let $\mathcal{A}$ be the default category, and let A and B be objects. Take D and $\mathcal{D}$ as suggested by $D\mathcal{D} = (\overset{A}{\bullet}\ \overset{B}{\bullet})$. Then a cocone δ for D is a two-member family $\delta = (f, g)$ with $f\colon A \to C$ and $g\colon B \to C$, where $C = \mathrm{tgt}\,\delta$.

Let $\gamma = (inl', inr')$ be a colimit for D. Then γ is a sum of A and B. To show this, we establish the existence of $\_\triangledown'\_$ for which $\triangledown'$-Charn holds, constructing $\triangledown'$ along the way. For arbitrary $f\colon A \to C$, $g\colon B \to C$, and morphism x,

$$
\begin{array}{ll}
 & inl' \,;\, x = f \ \text{ and } \ inr' \,;\, x = g \\
\equiv & \quad \text{composition of a cocone with a morphism, extensionality} \\
 & (inl', inr') \,;\, x = (f, g) \\
\equiv & \quad \gamma = (inl', inr') \text{ is colimit and } (f, g) \text{ is cocone for } D\text{; } \backslash\text{-Charn} \\
 & x = \gamma\backslash(f, g) \\
\equiv & \quad \textbf{define } f \triangledown' g = \gamma\backslash(f, g) \\
 & x = f \triangledown' g.
\end{array}
$$

Exercise: show that, if a sum of A and B exists, then there exists a colimit for the diagram D above.

2.42 Coequaliser as colimit. Let $\mathcal{A}$ be the default category, and let (f, g) be a parallel pair, with source A' and target A say. Take D and $\mathcal{D}$ as suggested in the top lines of the following pictures.

$$\begin{tikzcd} A' \arrow[rr, shift left, "f"] \arrow[rr, shift right, "g"'] \arrow[dr, "q'"'] & & A \arrow[dl, "q"] \\ & \bullet & \end{tikzcd} \qquad \begin{tikzcd} A' \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] \arrow[d, "p'"'] \arrow[dr, "q'"' near end] & A \arrow[dl, "p" near end] \arrow[d, "q"] \\ \bullet \arrow[r, "x"] & \bullet \end{tikzcd}$$

Then a cocone δ for D is a two-member family $\delta = (q', q)$ with $q'\colon A' \to C$ and $q\colon A \to C$, where $C = \operatorname{tgt}\delta$, and $q' = f \mathbin{;} q = g \mathbin{;} q$ (hence q' is fully determined by q alone).

Let $\gamma = (p', p)$ be a colimit for D; it may or may not exist. Then p is a coequaliser of (f, g). To show this, we establish the existence of a mapping $p\backslash\_$ for which coeq-Charn holds, constructing $p\backslash\_$ along the way. For arbitrary q with $f \mathbin{;} q = g \mathbin{;} q$, and morphism x,

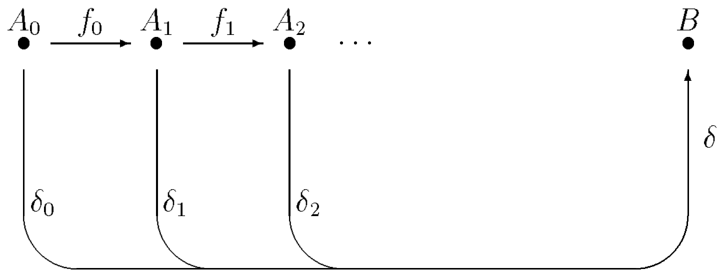
$$\begin{array}{ll} & p \mathbin{;} x = q \\ \equiv & \qquad \text{put } p' = f \mathbin{;} p = g \mathbin{;} p \text{ and } q' = f \mathbin{;} q = g \mathbin{;} q \\ & p' \mathbin{;} x = q' \text{ and } p \mathbin{;} x = q \\ \equiv & \qquad \text{composition of a cocone with a morphism, extensionality} \\ & (p', p) \mathbin{;} x = (q', q) \\ \equiv & \qquad \gamma = (p', p) \text{ is colimit and } (q', q) \text{ is cocone for } D;\ \backslash\text{-Charn} \\ & x = \gamma\backslash(q', q) \\ \equiv & \qquad \textbf{define } p\backslash q = \gamma\backslash(q', q) \text{ where } q' = f \mathbin{;} q = g \mathbin{;} q \\ & x = p\backslash q. \end{array}$$

Exercise: show that, if there exists a coequaliser for the parallel pair (f, g), then there exists a colimit for the diagram D above.

2.43 "Limit point" as colimit. This example explains the name 'limit': (co)limits may define real limiting points. Let $\mathcal{S}et$ be the default category, and consider an infinite sequence of sets, each including the previous one: $A_0 \subseteq A_1 \subseteq A_2 \ldots$. The subsets are partly identical (they have some elements in common), but categorically they are *different objects*. An inclusion $A \subseteq A'$ is expressed categorically by the existence of an injective function $f\colon A \to A'$ that *embeds* each element from A into A'. (In this way, a set may be a subset of another one in several distinct ways.) So the sequence of embeddings is expressed by the diagram:

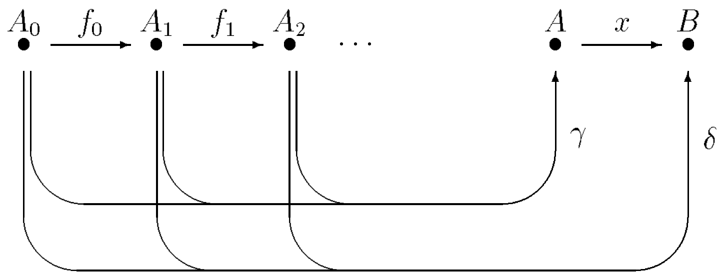
$$\begin{tikzcd} A_0 \arrow[r, "f_0"] & A_1 \arrow[r, "f_1"] & A_2 & \cdots \end{tikzcd}$$

Each composition $f_i \mathbin{;} f_{i+1} \mathbin{;} \cdots \mathbin{;} f_{j-1}\colon A_i \to A_j$ denotes the accumulated embedding of A_i into A_j. Now consider a cocone δ for that diagram:



The equalities $f_i \mathbin{;} \delta_{i+1} = \delta_i$ imply that $f_i \mathbin{;} f_{i+1} \mathbin{;} \cdots \mathbin{;} f_{j-1} \mathbin{;} \delta_j = \delta_i$. So each element from each of the subsets is mapped "unambiguously" into B via δ. (It is not true in general that B includes each A_i, since δ_i need not be injective; indeed, B might be the one-point set.)

Let $A = \bigcup(n :: A_n)$, the infinite union of all the subsets, and let γ_i be the embedding of A_i in A in such a way that each element of each of the subsets is mapped "unambiguously" (as explained for δ above) into A. We claim that γ is a colimit for that diagram.



To prove the claim, we must show that, for arbitrary cocone δ for that diagram, there is precisely one solution x for the equation $\gamma \mathbin{;} x = \delta$. Consider the function x' that maps an element $a \in A$ onto $\delta_i(a') \in B$, where i, a' are such that $\gamma_i(a') = a$. There exists for every $a \in A$ a pair i, a' with $\gamma_i(a') = a$, since by definition A is the *least* set including all A_i. The 'commutativity of the triangles' (of both γ and δ) implies that the specification of x' is unambiguous: $\delta_i(a') = \delta_j(a'')$ if $\gamma_i(a') = \gamma_j(a'')$. Clearly, this x' is a solution for x in $\gamma \mathbin{;} x = \delta$; we leave it as an exercise to show that it is the only solution.

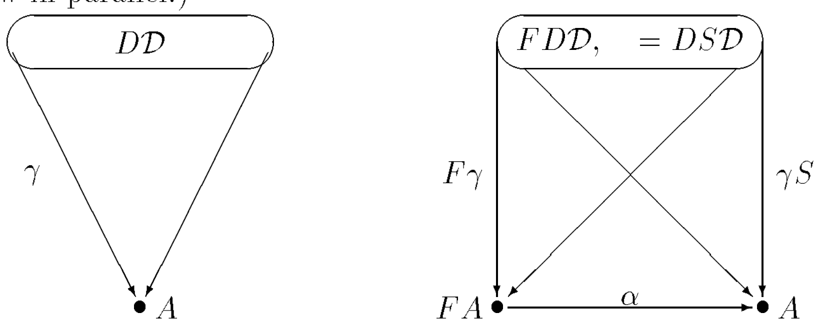
In effect, γ represents the infinite composition (embedding) $f_0 \mathbin{;} f_1 \mathbin{;} f_2 \mathbin{;} \cdots$; more precisely, γ is the *limit* of $f_0 \mathbin{;} f_1 \mathbin{;} f_2 \mathbin{;} \cdots$.

2.44 Application. We present the well-known construction of an initial F-algebra. Our interest is *solely* in the algebraic, calculational style of various subproofs, not in the outline of the main proof. For completeness we will briefly define the notion of algebra; without any explanation. So you may postpone reading this application until you know what algebras are good for. The construction will require that the category has an initial object and a colimit for each ω-chain, and that functor F preserves colimits of ω-chains; briefly: the category is an ω-category and F is ω-cocontinuous.

Here is the definition of the notion of algebra and the category of F-algebras. Let $\mathcal{C}$ be a category, the default one. Let $F\colon \mathcal{C} \to \mathcal{C}$ be a functor. The category $\mathcal{A}lg(F)$, built upon

$\mathcal{C}$, is defined as follows. An object in $\mathcal{A}lg(F)$ is: a morphism φ in $\mathcal{C}$ of type $FA \to A$, for some A. Let φ and ψ be objects in $\mathcal{A}lg(F)$; then a morphism from φ to ψ in $\mathcal{A}lg(F)$ is: a morphism f in $\mathcal{C}$ satisfying $\varphi \,;\, f = Ff \,;\, \psi$. (Actually, thus defined $\mathcal{A}lg(F)$ is a pre-category rather than a category.) The objects and morphisms in $\mathcal{A}lg(F)$ are called F-**algebra**s and F-**homomorphism**s, respectively. Using the laws for initiality, instantiated to $\mathcal{A}lg(F)$, one can easily show that if α: $FA \to A$ is initial in $\mathcal{A}lg(F)$, then α is an isomorphism, α: $FA \cong A$.

Let $\mathcal{C}$ be a category, the default one. Given endofunctor F we wish to construct an F-algebra, α: $FA \to A$ say, that is initial in $\mathcal{A}lg(F)$. Forgoing initiality for the time being, we derive a construction of an α: $FA \to A$ as follows. (Read the steps and their explanation below in parallel!)



α: $FA \to A$

(a) $\Leftarrow$ definition isomorphism

α: $FA \cong A$

(b) $\Leftarrow$ definition cocone morphism (taking $A = \mathrm{tgt}\gamma = \mathrm{tgt}\gamma S$)

α: $F\gamma \cong \gamma S$ in $\bigvee(FD) \quad \wedge \quad FD = DS$

(c) $\equiv$ $F\gamma$ is colimit for FD (taking $\alpha = F\gamma \backslash \gamma S$)

γS is colimit for $DS \quad \wedge \quad FD = DS$.

Step (a): this is motivated by the wish that α be initial in $\mathcal{A}lg(F)$, and so α will be an isomorphism; in other words, in view of the required initiality the step is *no* strengthening.
Step (b): here we merely decide that α, A come from a (co)limit construction; this is true for many categorical constructions. So we aim at α: $F\gamma \cong \ldots$, where γ is a colimit (which we *assume* to exist) for a diagram D yet to be defined. Since $F\gamma$ is a FD-cocone, there has to be another FD-cocone on the dots. To keep things simple, we aim at an FD-cocone constructed from γ, say γS, where S is an endofunctor on $\mathrm{src}D$. Since γS is evidently a DS-cocone, and must be an FD-cocone, it follows that $FD = DS$ is another requirement.
Step (c): the hint '$F\gamma$ is colimit for FD' follows from the *assumption* that F preserves colimits, and the definition $\alpha = F\gamma \backslash \gamma S$ is *forced* by (the proof of) the uniqueness of initial objects. (It is indeed very easy to verify that $F\gamma \backslash \gamma S$ and $\gamma S \backslash F\gamma$ are each other's inverse.)
We shall now complete the construction in the following three parts.

1. Construction of D, S such that $FD = DS$.
2. Proof of 'γS is colimit for DS' where γ is a colimit for D.
3. Proof of 'α is initial in $\mathcal{A}lg(F)$' where $\alpha = F\gamma\backslash\gamma S$.

Part 1. (Construction of D, S such that $FD = DS$.) The requirement $FD = DS$ says that FD is a 'subdiagram' of D. This is easily achieved by making D a *chain* of iterated F applications, as follows.

Let ω be the category with objects $0, 1, 2, \ldots$ and a unique arrow from i to j (denoted $i{\le}j$) for every $i \le j$. So ω is the shape of a **chain**. The zero and successor functors $0, S\colon \omega \to \omega$ are defined by $0(i{\le}j) = 0{\le}0$ and $S(i{\le}j) = (i{+}1){\le}(j{+}1)$.

Assume that $\mathcal{C}$ has an initial object, 0 say. Define the diagram $D\colon \omega \to \mathcal{C}$ by $D(i{\le}j) = F^i(\![F^{j-i}0]\!)$, where $(\![\_]\!)$ abbreviates $(\![0 \to \_]\!)_{\mathcal{C}}$. It is quite easy to show that D is a functor, that is, $D(i{\le}j \,;\, j{\le}k) = D(i{\le}j) \,;\, D(j{\le}k)$. It is also immediate that $FD = DS$, since for arbitrary morphism $i{\le}j$:

$$
\begin{aligned}
& FD(i{\le}j) \\
= \quad & FF^i(\![F^{j-i}0]\!) \\
= \quad & F^{i+1}(\![F^{(j+1)-(i+1)}0]\!) \\
= \quad & D((i{+}1){\le}(j{+}1)) \\
= \quad & DS(i{\le}j).
\end{aligned}
$$

Thanks to the form of ω, natural transformations of the form $\varepsilon\colon D \dot{\to} G$ (arbitrary G) can be defined by induction, that is, by defining

$$
\begin{aligned}
\varepsilon 0 \quad &: \quad D0 \dot{\to} G0 \qquad \text{or, equivalently,} \qquad \varepsilon 0\colon D0 \to G0 \\
\varepsilon S \quad &: \quad DS \dot{\to} GS\,.
\end{aligned}
$$

We shall use this form of definition in Part 2 and Part 3 below.

Assume that $\mathcal{C}$ has a colimit γ for diagram D.

Part 2. (Proof of 'γS is colimit for DS' where γ is a colimit for D.) Our task is to construct for arbitrary DS-cocone δ a morphism $(\![\gamma S \to \delta]\!)_{\bigvee(DS)}$ such that

$$(\spadesuit) \qquad \gamma S \,;\, x = \delta \quad \equiv \quad x = (\![\gamma S \to \delta]\!)_{\bigvee(DS)}\,.$$

Our guess is that $\gamma\backslash\varepsilon$ may be chosen for $(\![\gamma S \to \delta]\!)_{\bigvee(DS)}$ for some suitably chosen ε: $D \dot{\to} \underline{\text{tgt}\delta}$ that depends on δ. This guess is sufficient to start the proof of ($\spadesuit$); we shall derive a definition of ε (more specifically, for $\varepsilon 0$ and εS) along the way.

$$
\begin{aligned}
& x = \gamma\backslash\varepsilon \\
\equiv \quad & \quad \backslash\text{-Charn}
\end{aligned}
$$

$$\begin{array}{ll}
& \gamma \,;\, x = \varepsilon \\
\equiv & \quad \text{observation at the end of Part 1} \\
& (\gamma \,;\, x)0 = \varepsilon 0 \quad \wedge \quad (\gamma \,;\, x)S = \varepsilon S \\
\equiv & \quad \text{composition of a cocone with a morphism, extensionality} \\
& \gamma 0 \,;\, x = \varepsilon 0 \quad \wedge \quad \gamma S \,;\, x = \varepsilon S \\
\equiv & \quad \{\text{ aiming at the left hand side of } (\spadesuit) \text{ }\} \\
& \quad \textbf{define } \varepsilon S = \delta \text{ (noting that } \delta\colon DS \dot{\to} \underline{\mathrm{tgt}\delta} \;=\; DS \dot{\to} \underline{\mathrm{tgt}\delta\, S}\text{)} \\
& \gamma 0 \,;\, x = \varepsilon 0 \quad \wedge \quad \gamma S \,;\, x = \delta \\
(*) \qquad \equiv & \quad \text{define } \varepsilon 0 \text{ below such that } \gamma S \,;\, x = \delta \;\Rightarrow\; \gamma 0 \,;\, x = \varepsilon 0 \text{ for all } x \\
& \gamma S \,;\, x = \delta .
\end{array}$$

In order to define $\varepsilon 0$ satisfying the requirement derived at step $(*)$, we calculate

$$\begin{array}{ll}
& \gamma 0 \,;\, x \\
= & \quad \{\text{anticipating next steps, introduce an identity}\} \\
& \quad \text{(recall that } \mathrm{tgt}\gamma \text{ has been called } A\text{, so that } \gamma\colon D \dot{\to} \underline{A}\text{)} \\
& \gamma 0 \,;\, \underline{A}(0 \leq 1) \,;\, x \\
= & \quad \text{naturality } \gamma \text{ (`commutativity of the triangle')} \\
& D(0 \leq 1) \,;\, \gamma 1 \,;\, x \\
= & \quad \text{assumption } \gamma S \,;\, x = \delta \\
& D(0 \leq 1) \,;\, \delta 0
\end{array}$$

so that the requirement at step $(*)$ is fulfilled by **defining** $\varepsilon 0 = D(0 \leq 1) \,;\, \delta 0$.

Part 3. (Proof of 'α is initial in $\mathcal{A}lg(F)$' where $\alpha = F\gamma \backslash \gamma S$.) Put again $A = \mathrm{tgt}\alpha = \mathrm{tgt}\gamma$, so that $\gamma\colon D \dot{\to} \underline{A}$ and $\alpha\colon FA \to A$. Let $\varphi\colon FB \to B$ be arbitrary. We have to construct a morphism in $\mathcal{C}$ from A to B, denoted $(\!|\alpha \rightarrow \varphi|\!)_F$, such that

$$(\clubsuit) \qquad F\gamma \backslash \gamma S \,;\, x = Fx \,;\, \varphi \quad \equiv \quad x = (\!|\alpha \rightarrow \varphi|\!)_F .$$

Our guess is that the required morphism $(\!|\alpha \rightarrow \varphi|\!)_F$ can be written as $\gamma \backslash \delta$ for some suitably chosen D-cocone δ. This guess is sufficient to start the proof of $(\clubsuit)$, deriving a definition for δ (more specifically, for $\delta 0$ and δS) along the way:

$$\begin{array}{ll}
& F\gamma \backslash \gamma S \,;\, x = Fx \,;\, \varphi \\
\equiv & \quad \backslash\text{-Fusion} \\
& F\gamma \backslash (\gamma S \,;\, x) = Fx \,;\, \varphi \\
\equiv & \quad \backslash\text{-Charn}[\,\gamma, \delta, x := F\gamma,\ \gamma S \,;\, x,\ Fx \,;\, \varphi\,] \\
& F\gamma \,;\, Fx \,;\, \varphi = \gamma S \,;\, x \\
\equiv & \quad \text{lhs: functor, rhs: composition of cocone with a morphism}
\end{array}$$

$$
\begin{array}{lll}
 & & F(\gamma \,;\, x) \,;\, \varphi = (\gamma \,;\, x)S \\
(*) & \equiv & \quad \text{explained and proved below (defining } \delta \text{)} \\
 & & \gamma \,;\, x = \delta \\
 & \equiv & \quad \backslash\text{-Charn} \\
 & & x = \gamma\backslash\delta.
\end{array}
$$

Arriving at the line above $(*)$ I see no way to make progress except to work bottom-up from the last line. Having the lines above and below $(*)$ available, we define δSn in terms of δn by

$$\delta S \quad = \quad F\delta \,;\, \varphi\,,$$

a definition that is also suggested by type considerations alone. Now part $\Leftarrow$ of equivalence $(*)$ is immediate:

$$
\begin{array}{ll}
 & F(\gamma \,;\, x) \,;\, \varphi = (\gamma \,;\, x)S \\
\Leftarrow & \quad \text{definition } \delta S\text{: } F\delta \,;\, \varphi = \delta S \\
 & \gamma \,;\, x = \delta.
\end{array}
$$

For part $\Rightarrow$ of equivalence $(*)$ we argue as follows, assuming the line above $(*)$ as a premise, and defining $\delta 0$ along the way.

$$
\begin{array}{ll}
 & \gamma \,;\, x = \delta \\
\equiv & \quad \text{induction principle} \\
 & (\gamma \,;\, x)\mathit{0} = \delta\,\mathit{0} \quad \wedge \quad \forall(n :: \quad (\gamma \,;\, x)n = \delta n \;\Rightarrow\; (\gamma \,;\, x)Sn = \delta Sn) \\
\equiv & \quad \text{proved below: the `base' in (i), and the `induction step' in (ii)} \\
 & \mathit{true}.
\end{array}
$$

For (i), the induction base, we calculate:

$$
\begin{array}{ll}
 & \gamma 0 \,;\, x \\
= & \quad \text{init-Charn, using } \gamma 0\text{: } 0 \to A \\
 & (\!|A|\!)_{\mathcal{C}} \,;\, x \\
= & \quad \text{init-Fusion, using } x\text{: } A \to B \\
 & (\!|B|\!)_{\mathcal{C}} \\
= & \quad \textbf{define } \delta 0 = (\!|B|\!)_{\mathcal{C}} \\
 & \mathit{true}.
\end{array}
$$

And for (ii), the induction step, we calculate for arbitrary n, using the induction hypothesis $(\gamma \,;\, x)n = \delta n$,

$$
\begin{array}{ll}
 & (\gamma \,;\, x)Sn \\
= & \quad \text{line above } (*)
\end{array}
$$

$$
\begin{array}{ll}
 & (F(\gamma \mathbin{;} x) \mathbin{;} \varphi)n \\
= & \qquad \text{hypothesis } (\gamma \mathbin{;} x)n = \delta n \\
 & (F\delta \mathbin{;} \varphi)n \\
= & \qquad \text{definition } \delta S \\
 & (\delta S)n
\end{array}
$$

as desired. This completes the entire construction and proof.

Appendix A

More on adjointness

We give several equivalent definitions of adjointness, and some corollaries and theorems. [Note — added in proof: you'd better read the paper "Adjunctions" written by Fokkinga and Meertens, Draft version printed in December 1992.]

A.1 Global constants. Let $\mathcal{A}$ and $\mathcal{B}$ be categories, and let $F\colon \mathcal{A} \to \mathcal{B}$ and $G\colon \mathcal{B} \to \mathcal{A}$ be functors, fixed throughout the sequel.

A.2 Default typing. Unless stated otherwise, variables A', A, f, φ (all in $\mathcal{A}$) and B, B', g, ψ (all in $\mathcal{B}$) are arbitrary, and have the typing indicated below.

$$\begin{array}{lcl@{\qquad\qquad}lcl} A', A & \in & \text{Objects of } \mathcal{A} & B, B' & \in & \text{Objects of } \mathcal{B} \\ f & : & A' \to_{\mathcal{A}} A & g & : & B \to_{\mathcal{B}} B' \\ \varphi & : & A \to_{\mathcal{A}} GB & \psi & : & FA \to_{\mathcal{B}} B \end{array}$$

In addition, entities $\eta, \llbracket\ \rrbracket$ ("to $\mathcal{A}$") and $\varepsilon, \lceil\!\lceil\ \rceil\!\rceil$ ("to $\mathcal{B}$") depend on F, G and have the following typing.

$$\begin{array}{lcl@{\qquad\qquad}lcl} \eta_A & : & A \to_{\mathcal{A}} GFA & \varepsilon_B & : & FGB \to_{\mathcal{B}} B \end{array}$$

$$\frac{\psi\colon FA \to_{\mathcal{B}} B}{\llbracket\psi\rrbracket_{A,B}\colon A \to_{\mathcal{A}} GB} \qquad\qquad \frac{\varphi\colon A \to_{\mathcal{A}} GB}{\lceil\!\lceil\varphi\rceil\!\rceil_{A,B}\colon FA \to_{\mathcal{B}} B}$$

Mappings $\llbracket\_\rrbracket$ and $\lceil\!\lceil\_\rceil\!\rceil$ are called **lad** and **rad**, respectively, from *left adjungate* and *right adjungate*. As a memory aid: the first symbol of $\llbracket\ \rrbracket$ has the shape of an 'L' and therefore denotes *l*ad. In typewriter font I would write `lad( )` and `rad( )`. For readability I will omit the typing information whenever appropriate, as well as most subscripts. Omitting the subscripts is dangerous (and even erroneous) if categories $\mathcal{A}$ and $\mathcal{B}$ are built upon another one. For example, when B itself is a morphism in an underlying category, then ε_B might be a morphism that depends on (and is expressed in) B. Nevertheless, in the following calculations the subscripts are derivable from the context (in a mechanical way, like type inference in modern functional languages), thus justifying the omission; see paragraph 1.39.

A.3 Remark. The following theorem asserts the equivalence of several statements. Each of them defines "F is left adjoint to G".

So, in order to prove that F is left adjoint to G it suffices to establish just one of the statements, and when you know that F is left adjoint to G you may use *all* of the statements. Before we present the proof of the theorem, we also give some corollaries: additional properties of an adjunction.

A.4 Theorem. Statements Adjunction, Units, LadAdj, RadAdj, Fusions, and Charns are equivalent. Moreover, the various $\llbracket\ \rrbracket$ that are asserted to exist, can all be chosen equal; the same holds for $\lceil\!\lceil\ \rceil\!\rceil$, η, and ε.

Adjunction. There exist $\eta_{\_}$ and $\varepsilon_{\_}$ typed as in paragraph A.2 and satisfying

A.5	$\varphi = \eta \,;\, G\psi \quad\equiv\quad F\varphi \,;\, \varepsilon = \psi$	Adjunction

Units. There exist $\eta_{\_}$ and $\varepsilon_{\_}$ typed as in paragraph A.2 and satisfying

A.6	$\eta\colon\ I \dot{\to}_{\mathcal{A}} GF$	unit-Ntrf
A.7	$\varepsilon\colon\ FG \dot{\to}_{\mathcal{B}} I$	co-unit-Ntrf
A.8	$\eta \,;\, G\varepsilon = id$	unit-Inv
A.9	$F\eta \,;\, \varepsilon = id$	Inv-co-unit

LadAdj. There exist $\llbracket\ \rrbracket_{\_,\_}$ and $\varepsilon_{\_}$ typed as in paragraph A.2 and satisfying

A.10	$\varepsilon\colon\ FG \dot{\to} I$	co-unit-Ntrf
A.11	$F\varphi \,;\, \varepsilon = \psi \quad\equiv\quad \varphi = \llbracket\psi\rrbracket$	lad-Charn

RadAdj. There exist $\lceil\!\lceil\ \rceil\!\rceil_{\_,\_}$ and $\eta_{\_}$ typed as in paragraph A.2 and satisfying

A.12	$\eta\colon\ I \dot{\to} GF$	unit-Ntrf
A.13	$\varphi = \eta \,;\, G\psi \quad\equiv\quad \lceil\!\lceil\varphi\rceil\!\rceil = \psi$	rad-Charn

Fusions. There exist $\llbracket\_\rrbracket_{\_,\_}$ and $\lceil\!\lceil\_\rceil\!\rceil_{\_,\_}$ typed as in paragraph A.2 and satisfying

A.14	$\llbracket Ff \,;\, \psi \,;\, g\rrbracket \quad=\quad f \,;\, \llbracket\psi\rrbracket \,;\, Gg$	lad-Fusion
A.15	$\lceil\!\lceil f \,;\, \varphi \,;\, Gg\rceil\!\rceil \quad=\quad Ff \,;\, \lceil\!\lceil\varphi\rceil\!\rceil \,;\, g$	rad-Fusion
A.16	$\varphi = \llbracket\psi\rrbracket \quad=\quad \lceil\!\lceil\varphi\rceil\!\rceil = \psi$	Inverse

Charns. There exist $\llbracket\_\rrbracket_{\_,\_}$ and $\lceil\!\lceil\_\rceil\!\rceil_{\_,\_}$, and $\eta_{\_}, \varepsilon_{\_}$ typed as in A.2 and satisfying

A.17	$F\varphi \,;\, \varepsilon = \psi \quad\equiv\quad \varphi = \llbracket\psi\rrbracket$	lad-Charn
A.18	$\varphi = \eta \,;\, G\psi \quad\equiv\quad \lceil\!\lceil\varphi\rceil\!\rceil = \psi$	rad-Charn
A.19	$\varphi = \llbracket\psi\rrbracket \quad=\quad \lceil\!\lceil\varphi\rceil\!\rceil = \psi$	Inverse

A.20 Corollary. Let F be left adjoint to G via $\eta_{\_}, \varepsilon_{\_}, \llbracket\ \rrbracket_{\_,\_}, \ulcorner\!\!\ulcorner\ \urcorner\!\!\urcorner_{\_,\_}$. Then:

A.21	$\eta = \llbracket id \rrbracket$	unit-Def
A.22	$\llbracket \psi \rrbracket = \eta \,;\, G\psi$	lad-Def
A.23	$F\llbracket \psi \rrbracket \,;\, \varepsilon = \psi$	lad-Self
A.24	$F\varphi \,;\, \varepsilon = F\varphi' \,;\, \varepsilon \quad \Rightarrow \quad \varphi = \varphi'$	lad-Uniq
A.25	$\varepsilon = \ulcorner\!\!\ulcorner id \urcorner\!\!\urcorner$	co-unit-Def
A.26	$\ulcorner\!\!\ulcorner \varphi \urcorner\!\!\urcorner = F\varphi \,;\, \varepsilon$	rad-Def
A.27	$\eta \,;\, G\ulcorner\!\!\ulcorner \varphi \urcorner\!\!\urcorner = \varphi$	rad-Self
A.28	$\eta \,;\, G\psi = \eta \,;\, G\psi' \quad \Rightarrow \quad \psi = \psi'$	rad-Uniq

A.29 Discussion. A quick glance at the formulas of the Theorem and the Corollary reveals that the same subexpressions turn up over and over again. In particular, a definition for $\llbracket \_ \rrbracket$ and $\ulcorner\!\!\ulcorner \_ \urcorner\!\!\urcorner$ (see lad- and rad-Def) can be read off directly from Adjunction; it is then also immediate that $\llbracket \_ \rrbracket$ and $\ulcorner\!\!\ulcorner \_ \urcorner\!\!\urcorner$ are each other's inverse, as expressed by law Inverse. Also, the pattern of the left-hand side of unit-Inv is clearly recognizable in Adjunction. The left-hand sides of laws lad- and rad-Charn are the same as the two sides of Adjunction. (It seems to me that law Adjunction is in general the easiest to work with when deriving consequences of an adjunction.)

Another reading of Adjunction is this: there is precisely one solution for ψ in the left-hand side equation, namely the ψ given by the right-hand side equation; and, also, there is precisely one solution for φ in the right-hand side equation, namely the one given by the left-hand side equation. The uniqueness of the solutions is also expressed by laws lad- and rad-Charn separately, and the solutions themselves are given by $\llbracket \_ \rrbracket$ and $\ulcorner\!\!\ulcorner \_ \urcorner\!\!\urcorner$. (See also paragraph 2.7 that explains the trick of expressing uniqueness of solutions in a way that is suitable for calculation.)

Law unit-Inv asserts that η has a post-inverse, law Inv-co-unit asserts that ε has a pre-inverse, and lad-Uniq asserts a kind of monic-ness for ε, and rad-Uniq asserts a kind of epic-ness for η. Law lad-Self shows that the effect of $\llbracket \_ \rrbracket$ can be undone; indeed, the definition of $\ulcorner\!\!\ulcorner \_ \urcorner\!\!\urcorner$ follows the pattern of the left-hand side of lad-Self. The name 'Self' derives from the observation that $\llbracket \psi \rrbracket$ itself is a solution for φ in the ever recurring equation $F\varphi \,;\, \varepsilon = \psi$. That nomenclature is consistent with the nomenclature that we've proposed for the laws of initiality.

The names of the laws and the symbols $\llbracket\ \rrbracket$ and $\ulcorner\!\!\ulcorner\ \urcorner\!\!\urcorner$ are not standard in category theory.

A.31 Lemma.

$$[\text{Inverse}] \quad \Rightarrow \quad ([\text{lad-Fusion}] \;\equiv\; [\text{rad-Fusion}])\,.$$

So, to prove Fusions it suffices to establish Inverse and either lad-Fusion or rad-Fusion. The proof of the lemma is simple:

$$
\begin{array}{ll}
 & \llbracket Ff \mathbin{;} \psi \mathbin{;} g \rrbracket = f \mathbin{;} \llbracket \psi \rrbracket \mathbin{;} Gg \\
\equiv & \qquad \text{Inverse} \\
 & Ff \mathbin{;} \psi \mathbin{;} g = \llbracket f \mathbin{;} \llbracket \psi \rrbracket \mathbin{;} Gg \rrbracket \\
\equiv & \qquad \text{for ` } \Rightarrow \text{ ' substitute } \psi := \llbracket \varphi \rrbracket \text{ (hence by Inverse } \llbracket \psi \rrbracket = \varphi \text{), and} \\
 & \qquad \text{for ` } \Leftarrow \text{ ' substitute } \varphi := \llbracket \psi \rrbracket \text{ (hence by Inverse } \llbracket \varphi \rrbracket = \psi \text{)} \\
 & Ff \mathbin{;} \llbracket \varphi \rrbracket \mathbin{;} g = \llbracket f \mathbin{;} \varphi \mathbin{;} Gg \rrbracket .
\end{array}
$$

A.31 Proof of Theorem A.4. We prove the theorem by circular implication:

... Adjunction $\Rightarrow$ Units $\Rightarrow$ LadAdj $\Rightarrow$ Fusions $\Rightarrow$ RadAdj $\Rightarrow$ Charn $\Rightarrow$ ...

> We urge the readers to try and prove some of the implications themselves, before reading all of the proofs below. It is an excellent exercise to become familiar with the calculational properties of an adjunction.

A.32 Proof of Adjunction $\Rightarrow$ Units. We establish co-unit-Ntrf; and unit-Inv along the way at line $(*)$.

$$
\begin{array}{lll}
 & \varepsilon\colon\ FG \dot{\to}_{\mathcal{B}} I & \\
\equiv & \qquad \text{definition } \dot{\to}\text{:} & \\
 & \qquad \text{For all } g\colon\ B \to_{\mathcal{B}} B' & \\
 & FGg \mathbin{;} \varepsilon_{B'} = \varepsilon_B \mathbin{;} g & \\
\equiv & \qquad \text{Adjunction}[\varphi, \psi := Gg, (\varepsilon \mathbin{;} g)] \text{ (from right to left)} & \\
 & Gg = \eta \mathbin{;} G(\varepsilon \mathbin{;} g) & \\
\equiv & \qquad \text{functor} & \\
 & Gg = \eta \mathbin{;} G\varepsilon \mathbin{;} Gg & \\
\Leftarrow & \qquad \text{Leibniz} & \\
(*) & id = \eta \mathbin{;} G\varepsilon & \text{(unit-Inv)} \\
\equiv & \qquad \text{Adjunction}[\varphi, \psi := id, \varepsilon] \text{ (from left to right)} & \\
 & F id \mathbin{;} \varepsilon = \varepsilon & \\
\equiv & \qquad \text{functor, identity} & \\
 & \mathit{true}. &
\end{array}
$$

Similarly for unit-Ntrf and Inv-co-unit.

A.33 Proof of Units $\Rightarrow$ LadAdj. We establish the equivalence LadCharn by circular 'follows from', defining $[\![\_]\!]$ along the way:

$$
\begin{array}{ll}
 & F\varphi \mathbin{;} \varepsilon = \psi \\
\equiv & \qquad \text{Inv-co-unit} \\
 & F\varphi \mathbin{;} \varepsilon = F\eta \mathbin{;} \varepsilon \mathbin{;} \psi \\
\equiv & \qquad \text{co-unit-Ntrf} \\
 & F\varphi \mathbin{;} \varepsilon = F\eta \mathbin{;} FG\psi \mathbin{;} \varepsilon \\
\Leftarrow & \qquad \text{functor, Leibniz} \\
 & \varphi = \eta \mathbin{;} G\psi \qquad = [\![\psi]\!] \ \text{ by } \textbf{defining } [\![\psi]\!] = \eta \mathbin{;} G\psi \qquad\qquad \text{(right hand side)} \\
\equiv & \qquad \text{unit-Inv} \\
 & \varphi \mathbin{;} \eta \mathbin{;} G\varepsilon = \eta \mathbin{;} G\psi \\
\equiv & \qquad \text{unit-Ntrf} \\
 & \eta \mathbin{;} GF\varphi \mathbin{;} G\varepsilon = \eta \mathbin{;} G\psi \\
\Leftarrow & \qquad \text{functor, Leibniz} \\
 & F\varphi \mathbin{;} \varepsilon = \psi.
\end{array}
$$

Actually, the above calculation also shows Units $\Rightarrow$ Adjunction.

A.34 Proof of LadAdj $\Rightarrow$ Fusions. We establish lad-Fusion as follows:

$$
\begin{array}{ll}
 & [\![Ff \mathbin{;} \psi \mathbin{;} g]\!] = f \mathbin{;} [\![\psi]\!] \mathbin{;} Gg \\
\equiv & \qquad \text{lad-Charn } [\varphi, \psi := \text{rhs, lhs}] \\
 & Ff \mathbin{;} \psi \mathbin{;} g = F(f \mathbin{;} [\![\psi]\!] \mathbin{;} Gg) \mathbin{;} \varepsilon \\
\equiv & \qquad \text{functor, co-unit-Ntrf} \\
 & Ff \mathbin{;} \psi \mathbin{;} g = Ff \mathbin{;} F[\![\psi]\!] \mathbin{;} \varepsilon \mathbin{;} g \\
\Leftarrow & \qquad \text{Leibniz} \\
 & \psi = F[\![\psi]\!] \mathbin{;} \varepsilon \\
\equiv & \qquad \text{lad-Charn}[\,\varphi := [\![\psi]\!]\,] \\
 & [\![\psi]\!] = [\![\psi]\!] \\
\equiv & \qquad \text{equality} \\
 & \mathit{true}.
\end{array}
$$

We establish Inverse, defining $[\![\_]\!]$ along the way:

$$
\begin{array}{ll}
 & \varphi = [\![\psi]\!] \\
\equiv & \qquad \text{lad-Charn} \\
 & F\psi \mathbin{;} \varepsilon = \psi \\
\equiv & \qquad \textbf{define } [\![\psi]\!] = F\psi \mathbin{;} \varepsilon
\end{array}
$$

$$[\![\varphi]\!] = \psi.$$

Now rad-Fusion follows by Lemma A.30.

A.35 Proof of Fusions $\Rightarrow$ RadAdj. We establish rad-Charn, starting with the right-hand side, since that doesn't contain the unknown η, and defining η along the way:

$$\begin{array}{ll} & \psi = [\![\varphi]\!] \\ \equiv & \quad \text{Inverse} \\ & (\!|\psi|\!) = \varphi \\ \equiv & \quad \text{lad-Fusion} \\ & (\!|id|\!) \,;\, G\psi = \varphi \\ \equiv & \quad \textbf{define } \eta = (\!|id|\!) \\ & \eta \,;\, G\psi = \varphi. \end{array}$$

Now we establish unit-Ntrf:

$$\begin{array}{ll} & \eta\colon I \dot{\to} GF \\ \equiv & \quad \text{definition naturality} \\ & \quad \text{For all } f: \\ & f \,;\, \eta = \eta \,;\, GFf \\ \equiv & \quad \text{definition } \eta \text{ (derived above)} \\ & f \,;\, (\!|id|\!) = (\!|id|\!) \,;\, GFf \\ \equiv & \quad \text{lad-Fusion at both sides} \\ & (\!|Ff \,;\, id|\!) = (\!|id \,;\, Ff|\!) \\ \equiv & \quad \text{identity, equality} \\ & \mathit{true}. \end{array}$$

A.36 Proof of RadAdj $\Rightarrow$ Charns. First we establish Inverse, defining $(\!|\_|\!)$ along the way:

$$\begin{array}{ll} & [\![\varphi]\!] = \psi \\ \equiv & \quad \text{rad-Charn} \\ & \varphi = \eta \,;\, G\psi \\ \equiv & \quad \textbf{define } (\!|\psi|\!) = \eta \,;\, G\psi \\ & \varphi = (\!|\psi|\!). \end{array}$$

Next we establish lad-Charn, defining ε along the way:

$$\begin{array}{ll} & \varphi = (\!|\psi|\!) \\ \equiv & \quad \text{Inverse (just derived)} \end{array}$$

$$
\begin{array}{lll}
 & \lceil\!\lceil \varphi \rceil\!\rceil = \psi & \\
(*) & \equiv \qquad \text{rad-Fusion (see below)} & \\
 & F\varphi \,;\, \lceil\!\lceil id \rceil\!\rceil = \psi & \\
 & \equiv \qquad \textbf{define } \varepsilon = \lceil\!\lceil id \rceil\!\rceil & \\
 & F\varphi \,;\, \varepsilon = \psi . &
\end{array}
$$

In step $(*)$ we have used rad-Fusion. This law follows from RadAdj in the same way as lad-Fusion follows from LadAdj, see paragraph A.34.

A.37 Proof of Charns $\Rightarrow$ Adjunction.

$$
\begin{array}{l}
\varphi = \eta \,;\, G\psi \\
\equiv \qquad \text{rad-Charn} \\
\psi = \lceil\!\lceil \varphi \rceil\!\rceil \\
\equiv \qquad \text{Inverse} \\
\lfloor\!\lfloor \psi \rfloor\!\rfloor = \varphi \\
\equiv \qquad \text{lad-Charn} \\
F\varphi \,;\, \varepsilon = \psi .
\end{array}
$$

This completes the proof of Theorem A.4.

A.38 Proof of Corollary A.20. For unit-Def we argue:

$$
\begin{array}{l}
\eta = \lfloor\!\lfloor id \rfloor\!\rfloor \\
\equiv \qquad \text{lad-Charn} \\
F\eta \,;\, \varepsilon = id \\
\equiv \qquad \text{Inv-co-unit} \\
true .
\end{array}
$$

For rad-Def, and rad-Self at line $(*)$, we argue:

$$
\begin{array}{llr}
 & \lceil\!\lceil \varphi \rceil\!\rceil = F\varphi \,;\, \varepsilon & \\
 & \equiv \qquad \text{Adjunction}[\,\psi := \lceil\!\lceil \varphi \rceil\!\rceil\,] & \\
(*) & \varphi = \eta \,;\, G\lceil\!\lceil \varphi \rceil\!\rceil & \text{(rad-Self)} \\
 & \equiv \qquad \text{rad-Charn}[\,\psi := \lceil\!\lceil \varphi \rceil\!\rceil\,] & \\
 & \lceil\!\lceil \varphi \rceil\!\rceil = \lceil\!\lceil \varphi \rceil\!\rceil & \\
 & \equiv \qquad \text{equality} & \\
 & true . &
\end{array}
$$

For rad-Uniq we argue

$$
\begin{array}{l}
\varphi = \varphi' \\
\Leftarrow \qquad \text{logic}
\end{array}
$$

$$
\begin{array}{ll}
 & \varphi = \lceil\!\lceil \psi \rceil\!\rceil = \varphi' \qquad \text{for some } \psi \\
\equiv & \qquad \text{rad-Charn} \\
 & F\varphi \,;\, \varepsilon = \psi = F\varphi' \,;\, \varepsilon \qquad \text{for some } \psi \\
\equiv & \qquad \text{logic} \\
 & F\varphi \,;\, \varepsilon = F\varphi' \,;\, \varepsilon.
\end{array}
$$

The other parts are proved similarly.

A.39 Exercise. For each $\mathcal{X}, \mathcal{Y} \in \{$ Adjunction, LadAdj, RadAdj, Fusions, Charn, Units $\}$, see whether you can prove $\mathcal{X} \equiv \mathcal{Y}$ or $\mathcal{X} \Rightarrow \mathcal{Y}$ directly, without relying on Theorem A.4. There are a lot of possibilities!!

A.40 Exercise. Give alternative proofs for each of the corollaries. For example, law unit-Def may also be proved directly from Charn by reducing the obligation $\eta = \lfloor\!\lfloor id \rfloor\!\rfloor$ to *true* by applying 'functor and identity' (introducing Gid after η), rad-Charn[$\varphi, \psi := \lfloor\!\lfloor id \rfloor\!\rfloor, id$], and Inverse, in that order. Another possibility is to apply lad-Charn, Adjunction, 'functor and identity'. Yet another possibility is to reduce the obligation $\eta = \lfloor\!\lfloor id \rfloor\!\rfloor$ to *true* by applying Inverse, Charn, 'functor and identity'.

A.41 Exercise. Barr and Wells [2] present RadAdj as a definition of "F is adjoint to G", and they prove LadAdj as a proposition. Compare our calculational proof of LadAdj $\Rightarrow$ RadAdj with the two-and-a-half page proof of Barr and Wells (Proposition 12.2.2, containing eight diagrams).

A.42 Exercise. Derive the typing (and the subscripts to $\lfloor\!\lfloor\ \rfloor\!\rfloor$, $\lceil\!\lceil\ \rceil\!\rceil$, η, and ε) for each of the laws, following the procedure of paragraph 1.39.

A.43 Exercise. Let F be left-adjoint to G via η, ε and also via η, ε'. Prove that $\varepsilon = \varepsilon'$.

A.44 Exercise. Find F and G such that F is left-adjoint to G via η, ε as well as via η', ε' with $(\eta, \varepsilon) \neq (\eta', \varepsilon')$. (Hint: take $F = G = I$, and $\mathcal{A} = \mathcal{B} =$ a category with one object and two morphisms.) So an adjointness does not determine the unit and co-unit uniquely.

A.45 Exercise. Suppose that F and F' are both left-adjoint to G. Prove that $F \cong F'$ (in category $\mathcal{F}tr(\mathcal{A}, \mathcal{B})$). (Hint: first establish the existence of natural transformations κ: $F \dot{\to} F'$ and, by symmetry, κ': $F' \dot{\to} F$; then show that $\kappa \,;\, \kappa' = id$ and, by symmetry, $\kappa' \,;\, \kappa = id$.) Conclude that κ and κ' are, in general, not uniquely determined by F, F', G. (Hint: see Exercise A.44.)

Yet another formulation

Here is another, important, elegant and compact, formulation of "F is adjoint to G". We first need some notation.

A.46 Hom-functor, notation. For arbitrary category $\mathcal{C}$ we define the two-place mapping $(\_ \to \_)$ by:

$$\begin{array}{lcll} (C \to C') & = & \{h \text{ in } \mathcal{C} \mid \quad h\colon\ C \to_{\mathcal{C}} C'\} & \text{an object in } \mathcal{S}et \\ (h \to h') & = & \lambda\chi. \quad h \,;\, \chi \,;\, h' & \text{a morphism in } \mathcal{S}et \text{ typed} \\ & & (\text{tgt}\, h \to \text{src}\, h') \to_{\mathcal{S}et} (\text{src}\, h \to \text{tgt}\, h') & \end{array}$$

It follows that $(\_ \to \_)$ is a functor (contravariant in its first parameter, since $\text{src}\, h$ and $\text{tgt}\, h$ change place in the source and target type of $(h \to h')$):

$$(\_ \to \_) \quad : \quad \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{S}et\,.$$

This functor is called the **hom-functor**, and is usually written $Hom(\_,\_)$. Our notation is motivated by, amongst others, the observation that $h\colon\ C \to C'$ equivales $h \in (C \to C')$.

For bifunctor $\_ \oplus \_$ and functors F, G , we write $F \oplus G$ for the functor $x \mapsto Fx \oplus Gx$. Further, let

$$\begin{array}{lclcl} X & = & Exl_{\mathcal{A}^{op},\mathcal{B}} & : & \mathcal{A}^{op} \times \mathcal{B} \to \mathcal{A}^{op} \\ Y & = & Exr_{\mathcal{A}^{op},\mathcal{B}} & : & \mathcal{A}^{op} \times \mathcal{B} \to \mathcal{B} \end{array}$$

denote the obvious extraction functors.

With all this notation $(F\,X \to Y)$ and $(X \to G\,Y)$ are functors of type $\mathcal{A}^{op} \times \mathcal{B} \to \mathcal{S}et$ that satisfy the following equations:

$$\begin{array}{lclcl} (F\,X \to Y)(A, B) & = & (F\,A \to B) & = & \{g \text{ in } \mathcal{B} \mid \quad g\colon\ F\,A \to_{\mathcal{B}} B\} \\ (F\,X \to Y)(f, g) & = & (F\,f \to g) & = & \lambda\psi. \quad F f \,;\, \psi \,;\, g \end{array}$$

and

$$\begin{array}{lclcl} (X \to G\,Y)(A, B) & = & (A \to G\,B) & = & \{f \text{ in } \mathcal{A} \mid \quad f\colon\ A \to_{\mathcal{A}} G\,B\} \\ (X \to G\,Y)(f, g) & = & (f \to G\,g) & = & \lambda\varphi. \quad f \,;\, \varphi \,;\, Gg\,. \end{array}$$

Exercise: check the claims of the last sentence ('are functors', 'of type', 'satisfy the equations').

A.47 Theorem. The statement "F is left adjoint to G" is equivalent to IsoAdj.

IsoAdj.

A.48 $\quad (F\,X \to Y) \quad \cong \quad (X \to G\,Y) \qquad$ Iso

A.49 Proof of IsoAdj $\equiv$ Fusions. The isomorphism in Iso is apparently in the category where functors are the objects and natural transformations are the morphisms. So Iso abbreviates the following:

there exist natural transformations

A.50	$\lfloor\!\lfloor\rfloor\!\rfloor_{\_,\_}$	:	$(F\,X \to Y)$	$\dot\to$	$(X \to G\,Y)$	lad-Ntrf
A.51	$\lceil\!\lceil\rceil\!\rceil_{\_,\_}$	:	$(X \to G\,Y)$	$\dot\to$	$(F\,X \to Y)$	rad-Ntrf
A.52	that are each other's inverse.					Inverse

Now, law lad-Fusion is nothing but a detailed formulation of lad-Ntrf:

$$\begin{array}{ll}
& \lfloor\!\lfloor\rfloor\!\rfloor\colon\ (F\,X \to Y) \dot\to (X \to G\,Y) \\
\equiv & \quad \text{definition naturality} \\
& \quad \text{For all } (f,g)\colon (A,B) \to (A',B') \text{ in } \mathcal{A}^{op} \times \mathcal{B}: \\
& (F\,X \to Y)(f,g)\,;\,\lfloor\!\lfloor\rfloor\!\rfloor_{A',B'} = \lfloor\!\lfloor\rfloor\!\rfloor_{A,B}\,;\,(X \to G\,Y) \\
\equiv & \quad \text{property } (F\,X \to Y)(f,g) = (F f \to g) \text{ and similarly for } G \\
& (F f \to g)\,;\,\lfloor\!\lfloor\rfloor\!\rfloor_{A',B'} = \lfloor\!\lfloor\rfloor\!\rfloor_{A,B}\,;\,(f \to Gg) \\
\equiv & \quad \text{extensionality (in } \mathcal{S}et\text{)} \\
& \quad \text{For all } \psi \in (F A \to B): \\
& ((F f \to g)\,;\,\lfloor\!\lfloor\rfloor\!\rfloor_{A',B'})\psi = (\lfloor\!\lfloor\rfloor\!\rfloor_{A,B}\,;\,(f \to Gg))\psi \\
\equiv & \quad \text{composition applied: } (\mathcal{F}\,;\,\mathcal{G})x = \mathcal{G}(\mathcal{F}x) \\
& \lfloor\!\lfloor\rfloor\!\rfloor_{A',B'}\,((F f \to g)\psi) = (f \to Gg)\,(\lfloor\!\lfloor\rfloor\!\rfloor_{A,B}\psi) \\
\equiv & \quad \text{definition hom-functor } (_ \to _)\text{, writing } \lfloor\!\lfloor\rfloor\!\rfloor_{_,_}xyz \text{ as } \lfloor\!\lfloor xyz\rfloor\!\rfloor_{_,_} \\
& \lfloor\!\lfloor F f\,;\,\psi\,;\,g\rfloor\!\rfloor_{A',B'} = f\,;\,\lfloor\!\lfloor\psi\rfloor\!\rfloor_{A,B}\,;\,Gg.
\end{array}$$

Similarly for rad.

Initiality and colimit as adjointness

In this subsection we assume that you are familiar with the characterisation of initiality and colimits by laws init- and $\backslash$-Charn; see Section 2b and 2e.

A.53 Left-adjoints preserve initiality. Let $\mathcal{A},\mathcal{B}$ be arbitrary categories, and suppose that $\mathcal{A}$ has an initial object 0 and that $\mathcal{A},\mathcal{B},F,G,\eta,\varepsilon$ is an adjunction. We claim that $F0$ is initial in $\mathcal{B}$. To prove this, we establish the equivalence init-Charn $[f,A,B := g,F0,B]$ by circular implication, constructing $(\!|\_|\!)_{\mathcal{B}}$ along the way:

$$\begin{array}{ll}
& g\colon\ F0 \to_{\mathcal{B}} B \\
\Rightarrow & \quad \text{typing rules (composition, functor), } \eta\colon\ I \dot\to GF
\end{array}$$

$\eta_0 \,;\, Gg\colon\ 0 \rightarrow_{\mathcal{A}} GB$
$\equiv$ init-Charn $[f, A, B := (\eta_0 \,;\, Gg), 0, GB]$ in $\mathcal{A}$
$(\![GB]\!)_{\mathcal{A}} = \eta_0 \,;\, Gg$
$\equiv$ Adjunction $[\varphi, \psi := (\![GB]\!)_{\mathcal{A}}, g]$
$F(\![GB]\!)_{\mathcal{A}} \,;\, \varepsilon_B = g \qquad = (\![B]\!)_{\mathcal{B}}$ by **defining** $(\![B]\!)_{\mathcal{B}} = F(\![GB]\!)_{\mathcal{A}} \,;\, \varepsilon_B$
$\Rightarrow$ typing rules, $\varepsilon\colon\ FG \dot{\rightarrow} I$, and $(\![GB]\!)_{\mathcal{A}}\colon\ 0 \rightarrow_{\mathcal{A}} GB$
$g\colon\ F0 \rightarrow B$.

Exercise: is the first step also valid with $\equiv$ instead of $\Rightarrow$, thus shortening the proof?
Exercise: give an alternative proof, using $\llbracket\,\rrbracket$ and $\lceil\!\lceil\,\rceil\!\rceil$ and Inverse. Is there an essential difference between your proof and the one above?
Exercise: instantiate this proof to the case where $G = \underline{0}$, the constant functor mapping each B to 0 and each g to id_0. What is, in this case, $(\![\_]\!)_{\mathcal{B}}$?
Exercise: formulate the theorem as concise as possible for the special case that $\mathcal{A}$ is taken to be $\mathit{1}$, the category with one object and one morphism.

A.54 Initiality determines an adjunction. Let $\mathcal{B}$ be a category with an initial object 0. Then, for each category $\mathcal{A}$ with a final object 1 (for example, $\mathcal{A} = \mathit{1}$), there is an adjunction between $\mathcal{A}$ and $\mathcal{B}$.
Proof. Let F and G be the constant functors $F = \underline{0}\colon\ \mathcal{A} \rightarrow \mathcal{B}$ and $G = \underline{1}\colon\ \mathcal{B} \rightarrow \mathcal{A}$. We claim that F is left-adjoint to G. To prove this, we establish Adjunction, constructing η and ε along the way. For arbitrary A, B and $f\colon\ A \rightarrow_{\mathcal{A}} GB$ and $g\colon\ FA \rightarrow_{\mathcal{B}} B$,

$f = \eta_A \,;\, Gg \quad \equiv \quad Ff \,;\, \varepsilon_B = g$
$\equiv$ definition F and G, identity
$f = \eta_A \quad \equiv \quad \varepsilon_B = g$
$\equiv$ anticipating the next two steps, **define** $\eta_A = [\![A]\!]_{\mathcal{A}}$ and $\varepsilon_B = (\![B]\!)_{\mathcal{B}}$
$f = [\![A]\!] \quad \equiv \quad (\![B]\!) = g$
$\equiv$ final-Charn and init-Charn
$f\colon\ A \rightarrow 1 \quad \equiv \quad g\colon\ 0 \rightarrow B$
$\equiv$ typing f, g, and definition F, G
$\mathit{true} \quad \equiv \quad \mathit{true}.$

Actually, we have shown both sides of the equivalence to be true, rather than to be the same truth value. It still remains to prove that η and ε are natural; this is left as an exercise.
Exercise: show that $\mathcal{B}$ has an initial object if, and only if, there exists an adjunction between $\mathit{1}$ (the category with one object and one morphism) and $\mathcal{B}$.

A.55 Left-adjoints preserve colimits. Let $\mathcal{A}$ and $\mathcal{B}$ be arbitrary categories. Let $F\colon\ \mathcal{A} \rightarrow \mathcal{B}$ be left-adjoint to $G\colon\ \mathcal{B} \rightarrow \mathcal{A}$, and let $D\colon\ \mathcal{D} \rightarrow \mathcal{A}$ be a functor (a diagram in

$\mathcal{A}$). Suppose that γ is a colimit in $\mathcal{A}$ for D. Then $F\gamma$ is a colimit in $\mathcal{B}$ for FD.
Proof. First observe that functors preserve cocones: if δ is a cocone for D: $\mathcal{D} \to \mathcal{A}$, then $F\delta$ is a cocone for FD: $\mathcal{D} \to \mathcal{B}$. We claim that $F\gamma$ is a colimit for FD. To prove this, we establish colimit-Charn, constructing $F\gamma\backslash\_$ along the way. For arbitrary cocone δ in $\mathcal{B}$ for FD,

$$
\begin{array}{lll}
 & & F\gamma \,;\, x = \delta \\
 & \equiv & \qquad \text{composition of cocone with a morphism, extensionality} \\
 & & F\gamma_A \,;\, x = \delta_A \qquad \text{for each } A \text{ in } \mathcal{D} \\
 & \equiv & \qquad \text{Inverse, noting that both sides have type } FDA \to_{\mathcal{B}} \operatorname{tgt}\delta \\
 & & \llbracket F\gamma_A \,;\, x \rrbracket = \llbracket \delta_A \rrbracket \qquad \text{for each } A \text{ in } \mathcal{D} \\
 & \equiv & \qquad \text{lad-Fusion} \\
 & & \gamma_A \,;\, \llbracket x \rrbracket = \llbracket \delta_A \rrbracket \qquad \text{for each } A \text{ in } \mathcal{D} \\
(*) & \equiv & \qquad \text{for } \Rightarrow\text{: } \textbf{define } \delta' \text{ by } \delta'_A = \llbracket \delta_A \rrbracket \text{ for each } A \text{ in } \mathcal{D} \\
 & & \qquad \text{for } \Leftarrow\text{: note that by } (\star) \text{ we have } \delta'_A = \llbracket \delta_A \rrbracket \text{ for each } A \text{ in } \mathcal{D} \\
 & & \gamma \,;\, \llbracket x \rrbracket = \delta' \\
 & \equiv & \qquad \gamma \text{ is colimit for } D\text{, colimit-Charn} \\
 & & \llbracket x \rrbracket = \gamma\backslash\delta' \\
 & \equiv & \qquad \text{Inverse} \\
 & & x = \lceil\!\lceil \gamma\backslash\delta' \rceil\!\rceil \\
(\star) & \equiv & \qquad \textbf{define } F\gamma\backslash\delta = \lceil\!\lceil \gamma\backslash\delta' \rceil\!\rceil \text{ where } \delta'_A = \llbracket \delta_A \rrbracket\text{; observation below} \\
 & & x = F\gamma\backslash\delta.
\end{array}
$$

The definition of $F\gamma\backslash\_$ in step $(\star)$ requires some care. First, even though in general γ is not recoverable from $F\gamma$, here γ is known from the data of the theorem. Second, the notation $...\gamma\backslash\delta'...$ requires that δ' is a cocone for D, that is, δ': $D \dot{\to} \underline{X}$ for some object X in $\mathcal{A}$. It is almost trivial that δ' is a transformation from D to some $\underline{X}$; indeed, for arbitrary A in $\mathcal{D}$:

$$
\begin{array}{ll}
 & \delta'_A\colon\ DA \to_{\mathcal{A}} X \\
\Leftarrow & \qquad \text{definition } \delta'_A = \llbracket \delta_A \rrbracket\text{, typing } \llbracket\_\rrbracket \\
 & \delta_A\colon\ FDA \to_{\mathcal{B}} \operatorname{tgt}\delta \ \text{ and } \ X = G \operatorname{tgt}\delta \\
\Leftarrow & \qquad \text{assumption } \delta\colon\ FD \dot{\to} I\text{, } \textbf{define } X = G \operatorname{tgt}\delta \\
 & \mathit{true}.
\end{array}
$$

The verification of the naturality of δ' is less trivial:

$$
\begin{array}{ll}
 & \delta'\colon\ D \dot{\to} \underline{G \operatorname{tgt}\delta} \\
\equiv & \qquad \text{definition } \dot{\to}\text{;} \\
 & \qquad \text{For arbitrary } f\colon\ A \to_{\mathcal{D}} A'\text{:}
\end{array}
$$

$$
\begin{array}{ll}
 & Df \,;\, \delta'_{A'} = \delta'_{A} \,;\, id \\
\equiv & \quad \text{definition } \delta' \text{, identity} \\
 & Df \,;\, (\!|\delta_{A'}|\!) = (\!|\delta_{A}|\!) \\
\equiv & \quad \text{lad-Fusion} \\
 & (\!|FDf \,;\, \delta_{A'}|\!) = (\!|\delta_{A}|\!) \\
\Leftarrow & \quad \text{Leibniz, } \delta\colon FD \dot{\to} \underline{\mathrm{tgt}\,\delta} \\
 & \mathit{true}.
\end{array}
$$

Exercise: an alternative, much more abstract, proof might be obtained by considering colimits for D as initial objects in $\bigvee D$, and lifting functors to the cocone categories.
Exercise: check the proof for the case of the empty diagram.
Exercise: specialise this proof to sums, being particular colimits, and compare it with the proof by Barr and Wells [2] (Proposition 12.3.6, nine lines long).

Bibliography

[1] A. Asperti and G. Longo. *categories, Types, and Structures*. Foundations of Computing Series. The MIT Press, Cambridge, Ma, 1991.

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

[3] R.S. Bird. Lecture notes on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computing Science*. International Summer School directed by F.L. Bauer [et al.], Springer Verlag, 1989. NATO Advanced Science Institute Series (Series F: Computer and System Sciences Vol. 55).

[4] M.M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, University of Twente, dept Comp Sc, Enschede, The Netherlands, 1992.

[5] M.M. Fokkinga and E. Meijer. Program calculation properties of continuous algebras. Technical Report CS-R9104, CWI, Amsterdam, January 1991.

[6] C.A.R. Hoare. Notes on an Approach to Category Theory for Computer Scientists. In M. Broy, editor, *Constructive Methods in Computing Science*, pages 245–305. International Summer School directed by F.L. Bauer [et al.], Springer Verlag, 1989. NATO Advanced Science Institute Series (Series F: Computer and System Sciences Vol. 55).

[7] D.S. Scott. Relating theories of the lambda calculus. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, page 406. Academic Press, 1980.

Introduction to
"Law and Order in Algorithmics", Ph.D. Thesis by M. Fokkinga, Chapter 3: Algebras categorically, and Chapter 5: Datatypes without Signatures

There is a slight discrepancy between the notational conventions in the preceding chapters and my thesis [4]. In the thesis following conventions prevail.

- The default notation for objects is $a, b, c, \ldots$ instead of $A, B, C, \ldots$, and the default notation for 'the' initial and final object is $\mathit{0}$ and $\mathit{1}$, respectively. Variables x, y, z range over various entities, mostly morphisms but also objects. Variables φ, ψ, χ range over morphisms whose type has the form $Fa \to Ga$, for some object a (and given functors F, G).

Moreover:

- If $\dagger$ is a bifunctor (like $\times$ and $+$) and F, G are functors, then $F \dagger G$ denotes the functor defined by

  $$(F \dagger G)x \quad = \quad Fx \dagger Gx \quad \text{for all objects and morphisms } x\,.$$

  In particular, $I\!I = I \times I$; it maps each x onto $x \times x$. (Similarly, if the need arises, I would define $\mathit{2} = I + I$, so that $\mathit{2}x = x + x$ for each object and morphism x.)

- For product categories the extraction functors are denoted Exl, Exr while the symbol $\vartriangle$ also denotes the tupling (pairing) of functors.

- Juxtaposition associates to the *right*, so that $U\mu Fa = U(\mu(Fa))$, and binds **stronger** than any binary operation symbol, so that $Fa\dagger = (Fa)\dagger$. Binary operation symbol $;$ binds the **weakest** of all operation symbols in a term denoting a morphism. As usual, $\times$ has priority over $+$.

- For each object a, we use $\underline{a}$ to denote the **constant functor**:

  $$\begin{aligned} \underline{a}\,b &= a && \text{for each object } b \\ \underline{a}\,f &= id_a && \text{for each morphism } f\,. \end{aligned}$$

So, $\underline{1} + \underline{a} \times I$ is a functor, mapping each object b onto $1 + a \times b$ and mapping each morphism f onto $id_1 + id_a \times f$.

The same notation is also used for the constant function: $\underline{x}$ maps each argument onto x.

In the examples of Chapter 3 there occur references to paragraph 1.12, which introduces several datatypes informally. Here is a copy of that paragraph:

"Paragraph 1.12: Naturals, lists, streams". We shall frequently use naturals, cons lists, cons′ lists, and streams in examples, assuming that you know these concepts. Here is some informal explanation; the default category is $\mathcal{S}et$.

A distinguished one-element set is denoted 1. Function $!_a$: $a \to 1$ is the unique function from a to 1. Constants, like the number zero, will be modeled by functions with 1 as source, thus $zero$: $1 \to nat$. The sole member of 1 is sometimes written $()$, so that $zero() \in nat$ and $zero$ is called a *nullary* function.

For the **naturals** we use several known operations.

$$\begin{array}{lcll} zero & : & 1 \to nat & \text{zero, considered as a function from } 1 \\ succ & : & nat \to nat & \text{the successor function} \\ add & : & I\!I\, nat \to nat & \text{addition}. \end{array}$$

The set nat consists of all natural numbers. Functions on nat may be defined by induction on the $zero, succ$-structure of their argument.

For lists we distinguish between several variants.

The datatype of **cons lists** over a has as carrier the set La that consists of finite lists only. There are two functions nil and $cons$.

$$\begin{array}{lcl} nil & : & 1 \to La \\ cons & : & a \times La \to La\,. \end{array}$$

Depending on the context, nil and $cons$ are fixed for one specific set a, or they are considered to be polymorphic, that is, having the indicated type for each set a. In a very few cases a subscript will make this explicit. Each element from La can be written as a finite expression

$$cons(x_0,\ cons(x_1,\ \ldots\ cons(x_{n-1},\ nil)))\,.$$

So, functions over La can be defined by induction on the $nil, cons$ structure of their argument. For example, definitions of $size$: $La \to nat$ and $isempty$: $La \to La + La$ read

$$\begin{array}{lcl} nil \,;\, size & = & zero \\ cons \,;\, size & = & id \times size \,;\, add \end{array}$$

and

$$
\begin{array}{lcl}
nil \,;\, isempty & = & nil \,;\, inl \\
cons \,;\, isempty & = & cons \,;\, inr\,.
\end{array}
$$

Function $isempty$ sends its argument unaffected to the left/right component of its result type according to whether it is/isn't the empty list. A boolean result may be obtained by post-composing $isempty$ with $true \triangledown false$, see Section 2c for the *case* construct $\triangledown$. For each function $f\colon\ a \to b$ the so-called $map\, f$ for cons lists, denoted Lf, is defined by

$$
\begin{array}{lcl}
nil_a \,;\, Lf & = & nil_b \\
cons_a \,;\, Lf & = & f \times Lf \,;\, cons_b\,.
\end{array}
$$

If L were a functor, these equations assert that nil and $cons$ are natural transformations:

$$
\begin{array}{lcl}
nil & : & \underline{1} \dot{\to} L \\
cons & : & I \times L \dot{\to} L\,.
\end{array}
$$

We shall see that L really is a functor.

The datatype of **streams** over a has as carrier the set Sa that consists of infinite lists only. There are two functions to destruct a stream into a head in a and a tail that is a stream over a again.

$$
\begin{array}{lcl}
hd & : & Sa \to a \\
tl & : & Sa \to Sa\,.
\end{array}
$$

A function yielding a stream can be defined by inductively describing what its result is, in terms of applications of hd and tl. For example, the lists of naturals is defined as follows.

$$
\begin{array}{lcl}
from & : & nat \to S\, nat \\
from \,;\, hd & = & id \\
from \,;\, tl & = & succ \,;\, from \\
nats & : & 1 \to S\, nat \\
nats & = & zero \,;\, from
\end{array}
$$

By induction on n one can prove that

$$
nats \,;\, tl^n \,;\, hd \quad = \quad zero \,;\, succ^n\,.
$$

These functions act on infinite datastructures and the evaluation of $nats$ on a computing engine requires an infinite amount of time. Yet these functions are *total*; for each argument the result is well-defined. For each function $f\colon\ a \to b$ the so-called $map\, f$ for streams, denoted Sf, is defined by

$$
\begin{array}{lcl}
Sf \,;\, hd_b & = & hd_a \,;\, f \\
Sf \,;\, tl_b & = & tl_a \,;\, Sf\,.
\end{array}
$$

If S were a functor, these equations assert that hd and tl are natural transformations:

$$\begin{array}{lll} hd & : & S \dot{\to} I \\ tl & : & S \dot{\to} S\,. \end{array}$$

We shall see that S really is a functor.

The datatype of **cons′ lists** over a has as carrier the set $L'a$ that consists of all *finite and infinite* lists, called cons′ lists. There are several relevant functions.

$$\begin{array}{lll} nil' & : & 1 \to L'a \\ cons' & : & a \times L'a \to L'a \\ destruct' & : & L'a \to 1 + a \times L'a \\ isempty' & : & L'a \to L'a + L'a \end{array}$$

with

$$\begin{array}{lll} nil' \mathbin{;} destruct' & = & inl \\ cons' \mathbin{;} destruct' & = & inr \\ nil' \mathbin{;} isempty' & = & nil' \mathbin{;} inl \\ cons' \mathbin{;} isempty' & = & cons' \mathbin{;} inr\,. \end{array}$$

Since cons′ lists are possibly infinite, a ‘definition’ by induction on the $nil', cons'$-structure of cons′ lists is in general not possible; that would give *partially defined* functions, and these do not exist in our intended universe of discourse $\mathcal{S}et$. For example, consider the following equations with “unknown $size'$”.

$$\begin{array}{lll} nil' \mathbin{;} size' & = & zero \\ cons' \mathbin{;} size' & = & id \times size' \mathbin{;} add \end{array}$$

These do **not** define a total function $size'$: $L'a \to nat$, in contrast to the situation for cons lists. (Notice also the difference with the usual datatype of lists of nonstrict functional programming languages: next to finite and infinite lists, it comprises also partially defined lists.)