2

Coalgebras of Polynomial Functors

The previous chapter has introduced several examples of coalgebras and has illustrated basic coalgebraic notions such as behaviour and invariance (for those examples). This chapter will go deeper into the study of the area of coalgebra, introducing some basic notions, definitions and terminology. It will first discuss several fundamental set theoretic constructions, such as products, coproducts, exponents and powersets in a suitably abstract (categorical) language. These constructs are used to define a collection of elementary functors, the so-called polynomial functors. As will be shown in Section 2.2, this class of functors is rich enough to capture many examples of interesting coalgebras, including deterministic and non-deterministic automata. One of the attractive features of polynomial functors is that almost all of them have a final coalgebra – except when the (non-finite) powerset occurs. The unique map into a final coalgebra will appear as behaviour morphism, mapping a state to its behaviour. The two last sections of this chapter, Sections 2.4 and 2.5, provide additional background information, namely on algebras (as duals of coalgebras) and on adjunctions. The latter form a fundamental categorical notion describing back-and-forth translations that occur throughout mathematics.

2.1 Constructions on Sets

This section describes familiar constructions on sets, such as products, coproducts (disjoint unions), exponents and powersets. It does so in order to fix notation and to show that these operations are functorial, i.e. give rise to functors. This latter aspect is maybe not so familiar. Functoriality is essential for properly developing the theory of coalgebras; see Definition 1.4.5.

These basic constructions on sets are instances of more general constructions in categories. We shall give a perspective on these categorical

formulations, but we do not overemphasise this point. Readers without much familiarity with the theory of categories may then still follow the development, and readers who are quite comfortable with categories will recognise this wider perspective anyway.

Products

We recall that for two arbitrary sets X, Y the product $X \times Y$ is the set of pairs

$$X \times Y \;=\; \{(x, y) \mid x \in X \wedge y \in Y\}.$$

There are then obvious projection functions $\pi_1 \colon X \times Y \to X$ and $\pi_2 \colon X \times Y \to Y$ by $\pi_1(x, y) = x$ and $\pi_2(x, y) = y$. Also, for functions $f \colon Z \to X$ and $g \colon Z \to Y$ there is a tuple (or pairing) function $\langle f, g \rangle \colon Z \to X \times Y$ given by $\langle f, g \rangle(z) = (f(z), g(z)) \in X \times Y$ for $z \in Z$. Here are some basic equations which are useful in computations:

$$\begin{array}{ll} \pi_1 \circ \langle f, g \rangle \;=\; f & \qquad \langle \pi_1, \pi_2 \rangle \;=\; \mathrm{id}_{X \times Y} \\ \pi_2 \circ \langle f, g \rangle \;=\; g & \qquad \langle f, g \rangle \circ h \;=\; \langle f \circ h, g \circ h \rangle. \end{array} \tag{2.1}$$

The latter equation holds for functions $h \colon W \to Z$.

Given these equations it is not hard to see that the product operation gives rise to a bijective correspondence between pairs of functions $Z \to X$, $Z \to Y$ on the one hand, and functions $Z \to X \times Y$ into the product on the other. Indeed, given two functions $Z \to X$, $Z \to Y$, one can form their pair $Z \to X \times Y$. And in the reverse direction, given a function $Z \to X \times Y$, one can post-compose with the two projections π_1 and π_2 to get two functions $Z \to X$, $Z \to Y$. The above equations help show that these operations are each other's inverses. Such a bijective correspondence is conveniently expressed by a 'double rule', working in two directions:

$$\frac{Z \longrightarrow X \qquad Z \longrightarrow Y}{Z \longrightarrow X \times Y}\,. \tag{2.2}$$

Interestingly, the product operation $(X, Y) \mapsto X \times Y$ applies not only to sets, but also to functions: for functions $f \colon X \to X'$ and $g \colon Y \to Y'$ we can define a function $f \times g$, namely:

$$X \times Y \xrightarrow{\;f \times g\;} X' \times Y' \quad \text{given by} \quad (x, y) \longmapsto (f(x), g(y)). \tag{2.3}$$

Notice that the symbol $\times$ is overloaded: it is used both on sets and on functions. This product function $f \times g$ can also be described in terms of projections and

pairing as $f \times g = \langle f \circ \pi_1, g \circ \pi_2 \rangle$. It is easily verified that the operation $\times$ on functions satisfies

$$\mathrm{id}_X \times \mathrm{id}_Y = \mathrm{id}_{X \times Y} \quad \text{and} \quad (f \circ h) \times (g \circ k) = (f \times g) \circ (h \times k).$$

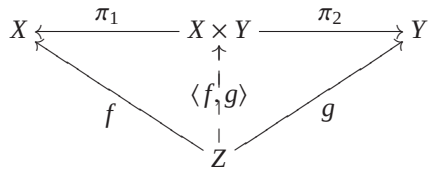
These equations express that the product $\times$ is *functorial*: it applies not only to sets, but also to functions, and it does so in such a way that identity maps and compositions are preserved (see Definition 1.4.3). The product operation $\times$ is a functor $\mathbf{Sets} \times \mathbf{Sets} \to \mathbf{Sets}$, from the product category $\mathbf{Sets} \times \mathbf{Sets}$ of $\mathbf{Sets}$ with itself, to $\mathbf{Sets}$.

Products of sets form an instance of the following general notion of product in a category.

Definition 2.1.1 Let $\mathbb{C}$ be a category. The **product** of two objects $X, Y \in \mathbb{C}$ is a new object $X \times Y \in \mathbb{C}$ with two projection morphisms

$$X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y,$$

which are universal: for each pair of maps $f\colon Z \to X$ and $g\colon Z \to Y$ in $\mathbb{C}$ there is a *unique* tuple morphism $\langle f, g \rangle\colon Z \to X \times Y$ in $\mathbb{C}$, making the following diagram commute.



The first two equations from (2.1) clearly hold for this abstract definition of product. The other two equations in (2.1) follow by using the uniqueness property of the tuple.

Products need not exist in a category, but if they exist they are determined up-to-isomorphism: if there is another object with projections $X \xleftarrow{p_1} X \otimes Y \xrightarrow{p_2} Y$ satisfying the above universal property, then there is a unique isomorphism $X \times Y \xrightarrow{\cong} X \otimes Y$ commuting with the projections. Similar results can be proven for the other constructs in this section.

What we have described is the product $X \times Y$ of *two* sets/objects X, Y. For a given X, we shall write $X^n = X \times \cdots \times X$ for the n-fold product (also known as **power**). The special case where $n = 0$ involves the empty product X^0, which is called a final or terminal object.

Definition 2.1.2 A **final object** in a category $\mathbb{C}$ is an object, usually written as $1 \in \mathbb{C}$, such that for each object $X \in \mathbb{C}$ there is a unique morphism $!_X\colon X \to 1$ in $\mathbb{C}$.

Not every category has a final object, but **Sets** does. Any singleton set is final. We choose one, and write it as $1 = \{*\}$. Notice then that elements of a set X can be identified with functions $1 \to X$. Hence we could forget about membership $\in$ and talk only about arrows.

When a category has binary products $\times$ and a final object 1, one says that the category has **finite products**: for each finite list $X_1, \ldots, X_n$ of objects one can form the product $X_1 \times \cdots \times X_n$. The precise bracketing in this expression is not relevant, because products are associative (up-to-isomorphism); see Exercise 2.1.8 below.

One can generalise these finite products to arbitrary, **set-indexed products**. For an index set I, and a collection $(X_i)_{i\in I}$ of I-indexed objects there is a notion of I-indexed product. It is an object $X = \prod_{i\in I} X_i$ with projections $\pi_i \colon X \to X_i$, for $i \in I$, which are universal as in Definition 2.1.1: for an arbitrary object Y and an I-indexed collection $f_i \colon Y \to X_i$ of morphisms there is a unique map $f = \langle f_i \rangle_{i\in I} \colon Y \to X$ with $\pi_i \circ f = f_i$, for each $i \in I$. In the category **Sets** such products exist and may be described as

$$\prod_{i\in I} X_i = \{t \colon I \to \bigcup_{i\in I} X_i \mid \forall i \in I.\, t(i) \in X_i\}. \tag{2.4}$$

Coproducts

The next construction we consider is the coproduct (or disjoint union, or sum) $+$. For sets X, Y we write their coproduct as $X + Y$. It is defined as

$$X + Y = \{(x, 1) \mid x \in X\} \cup \{(y, 2) \mid y \in Y\}.$$

The components 1 and 2 serve to force this union to be disjoint. These 'tags' enable us to recognise the elements of X and of Y inside $X + Y$. Instead of projections as above we now have 'coprojections' $\kappa_1 \colon X \to X+Y$ and $\kappa_2 \colon Y \to X + Y$ going in the other direction. One puts $\kappa_1(x) = (x, 1)$ and $\kappa_2(y) = (y, 2)$. And instead of tupling we now have 'cotupling' (sometimes called 'source tupling' or 'pattern matching'): for functions $f \colon X \to Z$ and $g \colon Y \to Z$ there is a cotuple function $[f, g] \colon X + Y \to Z$ going out of the coproduct, defined by case distinction:

$$[f, g](w) = \begin{cases} f(x) & \text{if } w = (x, 1) \\ g(y) & \text{if } w = (y, 2). \end{cases}$$

There are standard equations for coproducts, similar to those for products (2.1):

$$\begin{array}{ll} [f, g] \circ \kappa_1 = f & [\kappa_1, \kappa_2] = \mathrm{id}_{X+Y} \\ [f, g] \circ \kappa_2 = g & h \circ [f, g] = [h \circ f, h \circ g]. \end{array} \tag{2.5}$$

Earlier we described the essence of products in a bijective correspondence; see (2.2). There is a similar correspondence for coproducts, but with all arrows reversed:

$$\frac{X \longrightarrow Z \qquad Y \longrightarrow Z}{X + Y \longrightarrow Z}\,. \tag{2.6}$$

This duality between products and coproducts can be made precise in categorical language; see Exercise 2.1.3 below.

So far we have described the coproduct $X + Y$ on sets. We can extend it to functions in the following way. For $f\colon X \to X'$ and $g\colon Y \to Y'$ there is a function $f + g\colon X + Y \to X' + Y'$ by

$$(f + g)(w) = \begin{cases} (f(x), 1) & \text{if } w = (x, 1) \\ (g(y), 2) & \text{if } w = (y, 2). \end{cases} \tag{2.7}$$

Equivalently, we could have defined: $f + g = [\kappa_1 \circ f, \kappa_2 \circ g]$. This operation $+$ on functions preserves identities and composition:

$$\mathrm{id}_X + \mathrm{id}_Y = \mathrm{id}_{X+Y} \qquad \text{and} \qquad (f \circ h) + (g \circ k) = (f + g) \circ (h + k).$$

Thus, coproducts yield a functor $+\colon \mathbf{Sets} \times \mathbf{Sets} \to \mathbf{Sets}$, like products.

Coproducts in **Sets** satisfy some additional properties. For example, the coproduct is disjoint, in the sense that $\kappa_1(x) \neq \kappa_2(y)$, for all x, y. Also, the coprojections cover the coproduct: every element of a coproduct is of the form either $\kappa_1(x)$ or $\kappa_1(y)$. Further, products distribute over coproducts; see Exercise 2.1.7 below.

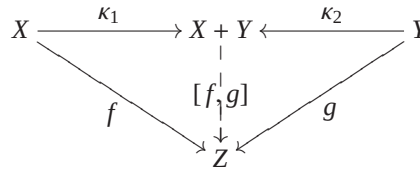
We should emphasise that a coproduct $+$ is very different from ordinary union $\cup$. For example, $\cup$ is idempotent: $X \cup X = X$, but there is not even an isomorphism between $X + X$ and X (if $X \neq \emptyset$). Union is an operation on subsets, whereas coproduct is an operation on sets.

Also the coproduct $+$ in **Sets** is an instance of a more general categorical notion of coproduct.

Definition 2.1.3 The **coproduct** of two objects X, Y in a category $\mathbb{C}$ is a new object $X + Y \in \mathbb{C}$ with two coprojection morphisms

$$X \xrightarrow{\kappa_1} X + Y \xleftarrow{\kappa_2} Y$$

satisfying a universal property: for each pair of maps $f\colon X \to Z$ and $g\colon Y \to Z$ in $\mathbb{C}$ there is a *unique* cotuple morphism $[f, g]\colon X + Y \to Z$ in $\mathbb{C}$, making the following diagram commute:



As for products, Equation (2.5) can be derived. Also, there is a notion of empty coproduct.

Definition 2.1.4 An **initial** object 0 in a category $\mathbb{C}$ has the property that for each object $X \in \mathbb{C}$ there a unique morphism $!_X \colon 0 \to X$ in $\mathbb{C}$.

The exclamation mark ! is often used to describe uniqueness, as in unique existence $\exists!$. Hence we use it to describe both maps $0 \to X$ out of initial objects and maps $X \to 1$ into final objects (see Definition 2.1.2). Usually this does not lead to confusion.

In **Sets** the empty set 0 is initial: for each set X there is precisely one function $0 \to X$, namely the empty function (the function with the empty graph). In **Sets** one has the additional property that each function $X \to 0$ is an isomorphism. This makes $0 \in$ **Sets** a so-called strict initial object.

As for products, one says that a category has finite coproducts when it has binary coproducts + together with an initial object 0. In that case one can form coproducts $X_1 + \cdots + X_n$ for any finite list of objects X_i. Taking the n-fold coproduct of the same object X yields what is called the **copower**, written as $n \cdot X = X + \cdots + X$. Also, a **set-indexed coproduct**, for a set I and a collection $(X_i)_{i \in I}$ of I-indexed objects, may exist. If so, it is an object $X = \coprod_{i \in I} X_i$ with coprojections $\kappa_i \colon X_i \to X$, for $i \in I$, which are universal: for an arbitrary object Y and a collection of maps $f_i \colon X_i \to Y$ there is a unique morphism $f = [f_i]_{i \in I} \colon \coprod_{i \in I} X_i \to Y$ with $f \circ \kappa_i = f_i$, for each $i \in I$. In the category **Sets** such coproducts are disjoint unions, like finite coproducts, but with tags from the index set I, as in

$$\coprod_{i \in I} X_i = \{(i, x) \mid i \in I \wedge x \in X_i\}. \tag{2.8}$$

Whereas products are well known, coproducts are relatively unfamiliar. But from a purely categorical perspective, they are essentially the same and not more difficult than products, because they are their duals (see Exercise 2.1.3 below). In a non-categorical setting however, the cotuple $[f, g]$ is a bit complicated, because it involves variable binding and pattern matching: in a term calculus one can write $[f, g](z)$ for instance as

$$\begin{array}{l}\text{CASES } z \text{ OF}\\ \quad \kappa_1(x) \longmapsto f(x)\\ \quad \kappa_2(y) \longmapsto g(y).\end{array}$$

Notice that the variables x and y are bound: they are mere place-holders, and their names are not relevant. Functional programmers are quite used to such cotuple definitions by pattern matching.

Another reason why coproducts are not so standard in mathematics is probably that in many algebraic structures coproducts coincide with products; in that case one speaks of biproducts. This is for instance the case for (commutative) monoids/groups and vector spaces and complete lattices; see Exercise 2.1.6. Additionally, in many continuous structures coproducts do not exist (e.g. in categories of domains).

However, within the theory of coalgebras coproducts play an important role. They occur in the construction of many functors F, used to describe coalgebras (namely as F-coalgebras; see Definition 1.4.5), in order to capture different output options, such as normal and abrupt termination in Section 1.1. But additionally, one can form new coalgebras from existing ones via coproducts. This will be illustrated next. It will be our first purely categorical construction. Therefore, it is elaborated in some detail.

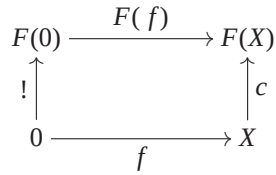
Proposition 2.1.5 *Let $\mathbb{C}$ be a category with finite coproducts $(0, +)$, and let F be an arbitrary endofunctor $\mathbb{C} \to \mathbb{C}$. The category* **CoAlg**(F) *of F-coalgebras then also has finite coproducts, given by*

$$\textit{initial coalgebra: } \begin{pmatrix} F(0) \\ \uparrow ! \\ 0 \end{pmatrix} \qquad \textit{coproduct coalgebra: } \begin{pmatrix} F(X+Y) \\ \uparrow \\ X+Y \end{pmatrix} \tag{2.9}$$

where the map $X + Y \to F(X+Y)$ on the right is the cotuple $[F(\kappa_1) \circ c, F(\kappa_2) \circ d]$, assuming coalgebras $X \xrightarrow{c} F(X)$ and $Y \xrightarrow{d} F(Y)$.

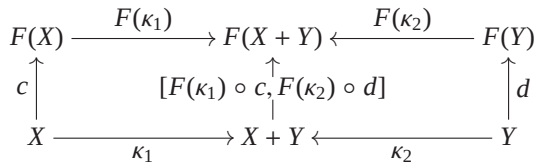
This result generalises to arbitrary (set-indexed) coproducts $\coprod_{i \in I} X_i$ (see Exercise 2.1.13) and also to coequalisers (see Exercise 2.1.14) (and thus to all colimits).

Proof It is important to distinguish between reasoning in the two different categories at hand, namely $\mathbb{C}$ and **CoAlg**(F). For the above map $0 \to F(0)$ to be an initial object in **CoAlg**(F) we have to show that there is a unique map to any object in **CoAlg**(F). This means, for an arbitrary coalgebra $c\colon X \to F(X)$ there must be a unique map f in **CoAlg**(F) of the form:

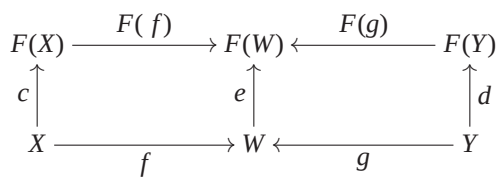


Since this map f must also be a map $0 \to X$ in $\mathbb{C}$, by initiality of $0 \in \mathbb{C}$, it can be only the unique map $f = !\colon 0 \to X$. We still have to show that for $f = !$ the above diagram commutes. But this follows again by initiality of 0: there can be only a single map $0 \to F(X)$ in $\mathbb{C}$. Hence both composites $c \circ f$ and $F(f) \circ !$ must be the same.

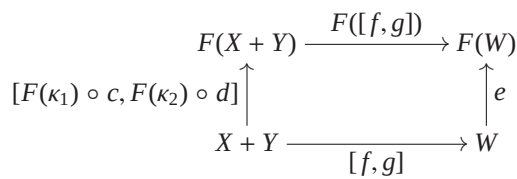
Next, in order to see that the coalgebra on the right in (2.9) is a coproduct in **CoAlg**(F), we precisely follow Definition 2.1.3. We have to have two coprojections in **CoAlg**(F), for which we take



It is almost immediately clear that these κ_1, κ_2 are indeed homomorphisms of coalgebras. Next, according to Definition 2.1.3 we must show that one can do cotupling in **CoAlg**(F). So assume two homomorphisms f, g of coalgebras:



These f, g are by definition also morphisms $X \to W, Y \to W$ in $\mathbb{C}$. Hence we can take their cotuple $[f, g]\colon X + Y \to W$ in $\mathbb{C}$, since by assumption $\mathbb{C}$ has coproducts. What we need to show is that this cotuple $[f, g]$ is also a map in **CoAlg**(F), in



This follows by using the coproduct equations (2.5):

$$\begin{aligned}
&F([f,g]) \circ [F(\kappa_1) \circ c,\ F(\kappa_2) \circ d] \\
&= [F([f,g]) \circ F(\kappa_1) \circ c,\ F([f,g]) \circ F(\kappa_2) \circ d] && \text{by (2.5)} \\
&= [F([f,g] \circ \kappa_1) \circ c,\ F([f,g] \circ \kappa_2) \circ d] && \text{since } F \text{ is a functor} \\
&= [F(f) \circ c,\ F(g) \circ d] && \text{by (2.5)} \\
&= [e \circ f,\ e \circ g] && f, g \text{ are coalgebra maps} \\
&= e \circ [f,g] && \text{see (2.5).}
\end{aligned}$$

Now we know that $[f,g]$ is a map in $\mathbf{CoAlg}(F)$. Clearly it satisfies $[f,g] \circ \kappa_1 = f$ and $[f,g] \circ \kappa_2 = g$ in $\mathbf{CoAlg}(F)$ because composition in $\mathbf{CoAlg}(F)$ is the same as in $\mathbb{C}$. Finally, Definition 2.1.3 requires that this $[f,g]$ is the unique map in $\mathbf{CoAlg}(F)$ with this property. But this follows because $[f,g]$ is the unique such map in $\mathbb{C}$. ❑

Thus, coproducts form an important construct in the setting of coalgebras.

Exponents

Given two sets X and Y one can consider the set

$$Y^X = \{f \mid f \text{ is a total function } X \to Y\}.$$

This set Y^X is sometimes called the function space, or exponent of X and Y. As in products and coproducts, it comes equipped with some basic operations. There is an evaluation function $\mathrm{ev}\colon Y^X \times X \to Y$, which sends the pair (f,x) to the function application $f(x)$. And for a function $f\colon Z \times X \to Y$ there is an abstraction function $\Lambda(f)\colon Z \to Y^X$, which maps $z \in Z$ to the function $x \mapsto f(z,x)$ that maps $x \in X$ to $f(z,x) \in Y$. Some basic equations are

$$\begin{aligned}
\mathrm{ev} \circ (\Lambda(f) \times \mathrm{id}_X) &= f \\
\Lambda(\mathrm{ev}) &= \mathrm{id}_{Y^X} \\
\Lambda(f) \circ h &= \Lambda(f \circ (h \times \mathrm{id}_X)).
\end{aligned} \tag{2.10}$$

Again, the essence of this construction can be summarised concisely in the form of a bijective correspondence, sometimes called currying.

$$\frac{Z \times X \longrightarrow Y}{Z \longrightarrow Y^X}\ . \tag{2.11}$$

Notice that if the set X is finite, say with n elements, then Y^X is isomorphic to the n-fold product $Y \times \cdots \times Y = Y^n$.

We have seen that both the product $\times$ and the coproduct $+$ give rise to functors $\mathbf{Sets} \times \mathbf{Sets} \to \mathbf{Sets}$. The situation for exponents is more subtle, because of the so-called contravariance in the first argument. This leads to an

exponent functor $\mathbf{Sets}^{\mathrm{op}} \times \mathbf{Sets} \to \mathbf{Sets}$, involving an opposite category for its first argument. We will show how this works.

For two maps $k\colon X \to U$ in $\mathbf{Sets}^{\mathrm{op}}$ and $h\colon Y \to V$ in $\mathbf{Sets}$ we need to define a function $h^k\colon Y^X \to V^U$ between exponents. The fact that $k\colon X \to U$ is a morphism in $\mathbf{Sets}^{\mathrm{op}}$ means that it really is a function $k\colon U \to X$. Therefore we can define h^k on a function $f \in Y^X$ as

$$h^k(f) = h \circ f \circ k. \tag{2.12}$$

This yields indeed a function in V^U. Functoriality also means that identities and compositions must be preserved. For identities this is easy:

$$(\mathrm{id}^{\mathrm{id}})(f) = \mathrm{id} \circ f \circ \mathrm{id} = f.$$

But for preservation of composition we have to remember that composition in an opposite category is reversed:

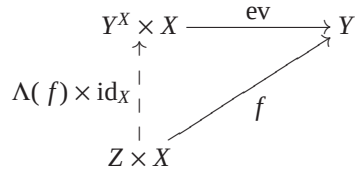
$$\begin{aligned}(h_2^{k_2} \circ h_1^{k_1})(f) = h_2^{k_2}(h_1^{k_1}(f)) &= h_2^{k_2}(h_1 \circ f \circ k_1)\\ &= h_2 \circ h_1 \circ f \circ k_1 \circ k_2\\ &= (h_2 \circ h_1)^{(k_1 \circ k_2)}(f)\\ &= (h_2 \circ_{\mathbb{C}} h_1)^{(k_2 \circ_{\mathbb{C}^{\mathrm{op}}} k_1)}(f).\end{aligned}$$

We conclude this discussion of exponents with the categorical formulation.

Definition 2.1.6 Let $\mathbb{C}$ be a category with products $\times$. The **exponent** of two objects $X, Y \in \mathbb{C}$ is a new object $Y^X \in \mathbb{C}$ with an evaluation morphism

$$Y^X \times X \xrightarrow{\ \mathrm{ev}\ } Y$$

such that for each map $f\colon Z \times X \to Y$ in $\mathbb{C}$ there is a *unique* abstraction morphism $\Lambda(f)\colon Z \to Y^X$ in $\mathbb{C}$, making the following diagram commute:



The following notions are often useful. A **cartesian closed category**, or **CCC** for short, is a category with finite products and exponents. And a **bicartesian closed category**, or **BiCCC**, is a CCC with finite coproducts. As we have seen, **Sets** is a BiCCC.

Powersets

For a set X we write $\mathcal{P}(X) = \{U \mid U \subseteq X\}$ for the set of (all) subsets of X. In more categorical style we shall also write $U \hookrightarrow X$ or $U \rightarrowtail X$ for $U \subseteq X$. These subsets will also be called predicates. Therefore, we sometimes write $U(x)$ for $x \in U$, and say in that case that U holds for x. The powerset $\mathcal{P}(X)$ is naturally ordered by inclusion: $U \subseteq V$ iff $\forall x \in X.\, x \in U \Rightarrow x \in V$. This yields a poset $(\mathcal{P}(X), \subseteq)$, with (arbitrary) meets given by intersection $\bigcap_{i\in I} U_i = \{x \in X \mid \forall i \in I.\, x \in U_i\}$, (arbitrary) joins by unions $\bigcup_{i\in I} U_i = \{x \in X \mid \exists i \in I.\, x \in U_i\}$ and negation by complement $\neg U = \{x \in X \mid x \notin U\}$. In brief, $(\mathcal{P}(X), \subseteq)$ is a complete Boolean algebra. Of special interest is the truth predicate $\top_X = (X \subseteq X)$, which always holds, and the falsity predicate $\bot_X = (\emptyset \subseteq X)$, which never holds. The two-element set $\{\bot, \top\}$ of Booleans is thus the powerset $\mathcal{P}(1)$ of the final object 1.

Relations may be seen as special cases of predicates. For example, a (binary) relation R on sets X and Y is a subset $R \subseteq X \times Y$ of the product set, i.e. an element of the powerset $\mathcal{P}(X \times Y)$. We shall use the following notations interchangeably:

$$R(x, y), \qquad (x, y) \in R, \qquad xRy.$$

Relations, like predicates, can be ordered by inclusion. The resulting poset $(\mathcal{P}(X \times Y), \subseteq)$ is again a complete Boolean algebra. It also contains a truth relation $\top_{X\times Y} \subseteq X \times Y$, which always holds, and a falsity relation $\bot_{X\times Y} \subseteq X \times Y$, which never holds.

Reversal and composition are two basic constructions on relations. For a relation $R \subseteq X \times Y$ we shall write $R^\dagger \subseteq Y \times X$ for the reverse relation given by $yR^\dagger x$ iff xRy. If we have another relation $S \subseteq Y \times Z$ we can describe the composition of relations $S \circ R$ as a new relation $(S \circ R) \subseteq X \times Z$, via $x(S \circ R)z$ iff $\exists y \in Y.\, R(x, y) \wedge S(y, z)$, as already described in (1.11).

Often we are interested in 'endo' relations $R \subseteq X \times X$ on a single set X. Of special interest then is the equality relation $\mathrm{Eq}(X) \subseteq X \times X$ given by $\mathrm{Eq}(X) = \{(x, y) \in X \times X \mid x = y\} = \{(x, x) \mid x \in X\}$. As we saw in Example 1.4.2.4, sets and relations form a category **SetsRel**, with equality relations as identity maps. The reversal operation $(-)^\dagger$ yields a functor $\mathbf{SetsRel}^{\mathrm{op}} \to \mathbf{SetsRel}$ that is the identity on objects and satisfies $R^{\dagger\dagger} = R$. It makes **SetsRel** into what is called a dagger category. Such categories are used for reversible computations, like in quantum computing; see e.g. [8].

The powerset operation $X \mapsto \mathcal{P}(X)$ is also functorial. For a function $f\colon X \to Y$ there is a function $\mathcal{P}(f)\colon \mathcal{P}(X) \to \mathcal{P}(Y)$ given by so-called **direct image**: for $U \subseteq X$,

$$\begin{aligned}\mathcal{P}(f)(U) &= \{f(x) \mid x \in U\} \\ &= \{y \in Y \mid \exists x \in X.\, f(x) = y \wedge x \in U\}.\end{aligned} \tag{2.13}$$

Alternative notation for this direct image is $f[U]$ or $\coprod_f(U)$. In this way we may describe the powerset as a functor $\mathcal{P}\colon \mathbf{Sets} \to \mathbf{Sets}$.

It turns out that one can also describe powerset as a functor $\mathcal{P}\colon \mathbf{Sets}^{\mathrm{op}} \to \mathbf{Sets}$ with the opposite of the category of sets as domain. In that case a function $f\colon X \to Y$ yields a map $f^{-1}\colon \mathcal{P}(Y) \to \mathcal{P}(X)$, which is commonly called **inverse image**: for $U \subseteq Y$,

$$f^{-1}(U) = \{x \mid f(x) \in U\}. \tag{2.14}$$

The powerset operation with this inverse image action on morphisms is sometimes called the contravariant powerset. But standardly we shall consider the 'covariant' powersets with direct images as functor $\mathbf{Sets} \to \mathbf{Sets}$. We shall frequently encounter these direct $\coprod_f$ and inverse f^{-1} images. They are related by a Galois connection:

$$\frac{\coprod_f(U) \subseteq V}{U \subseteq f^{-1}(V)}. \tag{2.15}$$

See also in Exercise 2.1.12 below.

We have seen bijective correspondences characterising products, coproducts, exponents and images. There is also such a correspondence for powersets:

$$\frac{X \longrightarrow \mathcal{P}(Y)}{\text{relations } \subseteq Y \times X}. \tag{2.16}$$

This leads to a more systematic description of a powerset as a so-called relation classifier. There is a special inhabitation $\in \subseteq Y \times \mathcal{P}(Y)$, given by $\in(y, U) \Leftrightarrow y \in U$. For any relation $R \subseteq Y \times X$ there is then a relation classifier, or characteristic function, $\mathsf{char}(R)\colon X \to \mathcal{P}(Y)$ mapping $x \in X$ to $\{y \in Y \mid R(y, x)\}$. This map $\mathsf{char}(R)$ is the unique function $f\colon X \to \mathcal{P}(Y)$ with $\in(y, f(x)) \Leftrightarrow R(y, x)$, i.e. with $(\mathrm{id} \times f)^{-1}(\in) = R$.

This formalisation of this special property in categorical language yields so-called power objects. The presence of such objects is a key feature of 'toposes'. The latter are categorical set-like universes, with constructive logic. They form a topic that goes beyond the introductory material covered in this text. The interested reader is referred to the extensive literature on toposes [279, 281, 166, 56, 346, 85].

Finally, we often need the *finite* powerset $\mathcal{P}_{\mathrm{fin}}(X) = \{U \in \mathcal{P}(X) \mid U \text{ is finite}\}$. It also forms a functor $\mathbf{Sets} \to \mathbf{Sets}$.

Injections and Surjections (in Sets)

A function $f\colon X \to Y$ is called injective (or an injection, a monomorphism or mono, for short, or a monic map) if $f(x) = f(x')$ implies $x = x'$. In that case we often write $f\colon X \rightarrowtail Y$ or $f\colon X \hookrightarrow Y$ in case X is a subset of Y. A surjection (or surjective function, epimorphism or epi or an epic map) is a map written as $f\colon X \twoheadrightarrow Y$ such that for each $y \in Y$ there is an $x \in X$ with $f(x) = y$. Injectivity and surjectivity can be formulated categorically (see Exercise 2.5.8 later on) and then appear as dual notions. In the category **Sets** these functions have some special 'splitting' properties that we shall describe explicitly because they are used from time to time.

The standard formulation of the axiom of choice (AC) says that for each collection $(X_i)_{i\in I}$ of non-empty sets there is a choice function $c\colon I \to \bigcup_{i\in I} X_i$ with $c(i) \in X_i$ for each $i \in I$. It is used for instance to see that the set-theoretic product $\prod_{i\in I} X_i$ from (2.4) is a non-empty set in case each X_i is non-empty.

An equivalent, more categorical, formulation of the axiom of choice is: every surjection $f\colon X \twoheadrightarrow Y$ has a section (also called splitting), a function $s\colon Y \to X$ in the reverse direction with $f \circ s = \mathrm{id}$. This s thus chooses an element $s(y)$ in the non-empty set $f^{-1}(y) = \{x \in X \mid f(x) = y\}$. Notice that such a section is an injection.

For injections there is a comparable splitting result. Assume $f\colon X \rightarrowtail Y$ in **Sets**, where $X \neq \emptyset$. Then there is a function $g\colon Y \to X$ with $g \circ f = \mathrm{id}$. This g is obtained as follows. Since $X \neq \emptyset$ we may assume an element $x_0 \in X$, and use it in

$$g(y) = \begin{cases} x & \text{if there is a (necessarily unique) element } x \text{ with } f(x) = y \\ x_0 & \text{otherwise.} \end{cases}$$

Notice that this g is a surjection $Y \twoheadrightarrow X$.

These observations will often be used in the following form.

Lemma 2.1.7 *Let* $F\colon$ **Sets** $\to$ **Sets** *be an arbitrary functor.*

1. *If* $f\colon X \to Y$ *is surjective, then so is* $F(f)\colon F(X) \to F(Y)$.
2. *If* $f\colon X \to Y$ *is injective and* X *is non-empty, then* $F(f)$ *is also injective.*

Proof 1. If $f\colon X \to Y$ is surjective, then, by the axiom of choice, there is a splitting $s\colon Y \to X$ with $f \circ s = \mathrm{id}_Y$. Hence $F(f) \circ F(s) = F(f \circ s) = F(\mathrm{id}_Y) = \mathrm{id}_{F(Y)}$. Thus $F(f)$ has a splitting (right inverse) and is thus surjective.

2. In the same way, as we have seen, for injective functions $f\colon X \to Y$ with $X \neq \emptyset$, there is a $g\colon Y \to X$ with $g \circ f = \mathrm{id}_X$. Thus $F(g) \circ F(f) = \mathrm{id}_{F(X)}$, so that $F(f)$ is injective. ❑

Exercises

2.1.1 Verify in detail the bijective correspondences (2.2), (2.6), (2.11) and (2.16).

2.1.2 Consider a poset $(D, \leq)$ as a category. Check that the product of two elements $d, e \in D$, if it exists, is the meet $d \wedge e$. And a coproduct of d, e, if it exists, is the join $d \vee e$.

Similarly, show that a final object is a top element $\top$ (with $d \leq \top$, for all $d \in D$) and that an initial object is a bottom element $\bot$ (with $\bot \leq d$, for all $d \in D$).

2.1.3 Check that a product in a category $\mathbb{C}$ is the same as a coproduct in $\mathbb{C}^{\mathrm{op}}$.

2.1.4 Fix a set A and prove that assignments $X \mapsto A \times X$, $X \mapsto A + X$ and $X \mapsto X^A$ are functorial and give rise to functors **Sets** $\to$ **Sets**.

2.1.5 Prove that the category **PoSets** of partially ordered sets and monotone functions is a BiCCC. The definitions on the underlying sets X of a poset $(X, \leq)$ are like for ordinary sets but should be equipped with appropriate orders.

2.1.6 Consider the category **Mon** of monoids with monoid homomorphisms between them.

1. Check that the singleton monoid 1 is both an initial and a final object in **Mon**; this is called a zero object.
2. Given two monoids $(M_1, +_1, 0_1)$ and $(M_2, +_2, 0_2)$, one defines a product monoid $M_1 \times M_2$ with componentwise addition $(x, y) + (x', y') = (x +_1 x', y +_2 y')$ and unit $(0_1, 0_2)$. Prove that $M_1 \times M_2$ is again a monoid, which forms a product in the category **Mon** with the standard projection maps $M_1 \xleftarrow{\pi_1} M_1 \times M_2 \xrightarrow{\pi_2} M_2$.
3. Note that there are also coprojections $M_1 \xrightarrow{\kappa_1} M_1 \times M_2 \xleftarrow{\kappa_2} M_2$, given by $\kappa_1(x) = (x, 0_2)$ and $\kappa_2(y) = (0_1, y)$, which are monoid homomorphisms and which make $M_1 \times M_2$ at the same time the coproduct of M_1 and M_2 in **Mon** (and hence a biproduct). *Hint*: Define the cotuple $[f, g]$ as $x \mapsto f(x) + g(x)$.

2.1.7 Show that in **Sets** products distribute over coproducts, in the sense that the canonical maps

$$(X \times Y) + (X \times Z) \xrightarrow{[\mathrm{id}_X \times \kappa_1, \mathrm{id}_X \times \kappa_2]} X \times (Y + Z)$$

$$0 \xrightarrow{\quad ! \quad} X \times 0$$

are isomorphisms. Categories in which this is the case are called **distributive**; see [101] for more information on distributive

categories in general and see [182] for an investigation of such distributivities in categories of coalgebras.

2.1.8 1. Consider a category with finite products $(\times, 1)$. Prove that there are isomorphisms:

$$X \times Y \cong Y \times X, \quad (X \times Y) \times Z \cong X \times (Y \times Z), \quad 1 \times X \cong X.$$

2. Similarly, show that in a category with finite coproducts $(+, 0)$ one has

$$X + Y \cong Y + X, \quad (X + Y) + Z \cong X + (Y + Z), \quad 0 + X \cong X.$$

(This means that both the finite product and coproduct structure in a category yield so-called *symmetric monoidal* structure. See [344, 85] for more information.)

3. Next, assume that our category also has exponents. Prove that

$$X^0 \cong 1, \qquad X^1 \cong X, \qquad 1^X \cong 1.$$

And also that

$$Z^{X+Y} \cong Z^X \times Z^Y, \quad Z^{X \times Y} \cong (Z^Y)^X, \quad (X \times Y)^Z \cong X^Z \times Y^Z.$$

2.1.9 Show that the finite powerset also forms a functor $\mathcal{P}_{\text{fin}} \colon \mathbf{Sets} \to \mathbf{Sets}$.

2.1.10 Check that

$$\mathcal{P}(0) \cong 1, \qquad \mathcal{P}(X + Y) \cong \mathcal{P}(X) \times \mathcal{P}(Y).$$

And similarly for the finite powerset $\mathcal{P}_{\text{fin}}$ instead of $\mathcal{P}$. This property says that $\mathcal{P}$ and $\mathcal{P}_{\text{fin}}$ are 'additive'; see [111].

2.1.11 Notice that a powerset $\mathcal{P}(X)$ can also be understood as exponent 2^X, where $2 = \{0, 1\}$. Check that the exponent functoriality gives rise to the contravariant powerset $\mathbf{Sets}^{\text{op}} \to \mathbf{Sets}$.

2.1.12 Consider a function $f \colon X \to Y$. Prove that

1. The direct image $\mathcal{P}(f) = \coprod_f \colon \mathcal{P}(X) \to \mathcal{P}(Y)$ preserves all joins and that the inverse image $f^{-1}(-) \colon \mathcal{P}(Y) \to \mathcal{P}(X)$ preserves not only joins but also meets and negation (i.e. all the Boolean structure).
2. There is a Galois connection $\coprod_f(U) \subseteq V \iff U \subseteq f^{-1}(V)$, as claimed in (2.15).
3. There is a product function $\prod_f \colon \mathcal{P}(X) \to \mathcal{P}(Y)$ given by $\prod_f(U) = \{y \in Y \mid \forall x \in X.\, f(x) = y \Rightarrow x \in U\}$, with a Galois connection $f^{-1}(V) \subseteq U \iff V \subseteq \prod_f(U)$.

2.1.13 Assume a category $\mathbb{C}$ has arbitrary, set-indexed coproducts $\coprod_{i\in I} X_i$. Demonstrate, as in the proof of Proposition 2.1.5, that the category **CoAlg**(F) of coalgebras of a functor $F\colon \mathbb{C} \to \mathbb{C}$ then also has such coproducts.

2.1.14 For two parallel maps $f, g\colon X \to Y$ between objects X, Y in an arbitrary category $\mathbb{C}$ a **coequaliser** $q\colon Y \to Q$ is a map in a diagram

$$X \underset{g}{\overset{f}{\rightrightarrows}} Y \xrightarrow{\ q\ } Q$$

with $q \circ f = q \circ g$ in a 'universal way': for an arbitrary map $h\colon Y \to Z$ with $h \circ f = h \circ g$ there is a unique map $k\colon Q \to Z$ with $k \circ q = h$.

1. An **equaliser** in a category $\mathbb{C}$ is a coequaliser in $\mathbb{C}^{\mathrm{op}}$. Formulate explicitly what an equaliser of two parallel maps is.
2. Check that in the category **Sets** the set Q can be defined as the quotient Y/R, where $R \subseteq Y \times Y$ is the least equivalence relation containing all pairs $(f(x), g(x))$ for $x \in X$.
3. Returning to the general case, assume a category $\mathbb{C}$ has coequalisers. Prove that for an arbitrary functor $F\colon \mathbb{C} \to \mathbb{C}$ the associated category of coalgebras **CoAlg**(F) also has coequalisers, as in $\mathbb{C}$: for two parallel homomorphisms $f, g\colon X \to Y$ between coalgebras $c\colon X \to F(X)$ and $d\colon Y \to F(Y)$ there is by universality an induced coalgebra structure $Q \to F(Q)$ on the coequaliser Q of the underlying maps f, g, yielding a diagram of coalgebras

$$\begin{pmatrix} F(X) \\ \uparrow c \\ X \end{pmatrix} \underset{g}{\overset{f}{\rightrightarrows}} \begin{pmatrix} F(Y) \\ \uparrow d \\ Y \end{pmatrix} \xrightarrow{\ q\ } \begin{pmatrix} F(Q) \\ \uparrow \\ Q \end{pmatrix}$$

with the appropriate universal property in **CoAlg**(F): for each coalgebra $e\colon Z \to F(Z)$ with homomorphism $h\colon Y \to Z$ satisfying $h \circ f = h \circ g$ there is a unique homomorphism of coalgebras $k\colon Q \to Z$ with $k \circ q = h$.

2.2 Polynomial Functors and Their Coalgebras

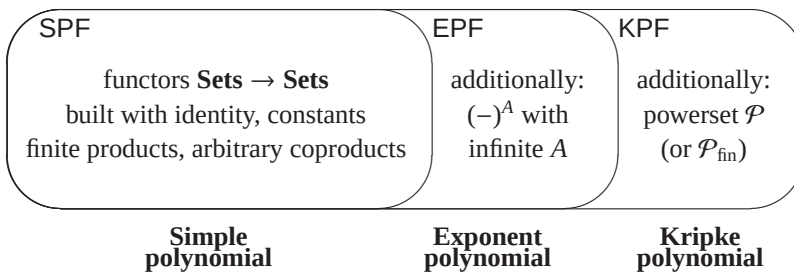
Earlier in Definition 1.4.5 we have seen the general notion of a coalgebra as a map $X \to F(X)$ in a category $\mathbb{C}$, where F is a functor $\mathbb{C} \to \mathbb{C}$. Here, in this section and in much of the rest of this text, we shall concentrate on a more restricted situation: as category $\mathbb{C}$ we use the category **Sets** of ordinary sets and functions. And as functors $F\colon$ **Sets** $\to$ **Sets** we shall use so-called polynomial

functors, such as $F(X) = A + (B \times X)^C$. These are functors built up inductively from certain simple basic functors, using products, coproducts, exponents and powersets for forming new functors. There are three reasons for this restriction to polynomial functors.

1. Polynomial functors are concrete and easy to grasp.
2. Coalgebras of polynomial functors include many of the basic examples; they suffice for the time being.
3. Polynomial functors allow definitions by induction, for many of the notions that we shall be interested in – notably relation lifting and predicate lifting in the next two chapters. These inductive definitions are easy to use and can be introduced without any categorical machinery.

This section contains the definition of polynomial functor and also many examples of such functors and of their coalgebras.

Definition 2.2.1 We define three collections of functors **Sets** $\to$ **Sets**, namely SPF, EPF, and KPF, for *simple*, *exponent* and *Kripke* polynomial functors, as in:



1. The collection SPF of **simple polynomial functors** is the least class of functors **Sets** $\to$ **Sets** satisfying the following four clauses.
   - i The identity functor **Sets** $\to$ **Sets** is in SPF.
   - ii For each set A, the constant functor $A\colon$ **Sets** $\to$ **Sets** is in SPF. Recall that it maps every set X to A and every function f to the identity id_A on A.
   - iii If both F and G are in SPF, then so is the product functor $F \times G$, defined as $X \mapsto F(X) \times G(X)$. On functions it is defined as $f \mapsto F(f) \times G(f)$; see (2.3).
   - iv If we have a non-empty set I and an I-indexed collection of functors F_i in SPF, then the set-indexed coproduct $\coprod_{i \in I} F_i$ is also in SPF. This new functor maps a set X to the I-indexed coproduct $\coprod_{i \in I} F_i(X) = \{(i, u) \mid i \in I \wedge u \in F_i(X)\}$. It maps a function $f\colon X \to Y$ to the mapping $(i, u) \mapsto (i, F_i(f)(u))$.

2. The collection EPF of **exponent polynomial functors** has the above four clauses, and additionally:

   v. For each set A, if F in EPF, then so is the 'constant' exponent F^A defined as $X \mapsto F(X)^A$. It sends a function $f\colon X \to Y$ to the function $F(f)^A = F(f)^{\mathrm{id}_A}$, which maps $h\colon A \to F(X)$ to $F(f) \circ h\colon A \to F(Y)$; see (2.12).

3. The class KPF of **Kripke polynomial functors** is the superset of SPF defined by the above clauses (i)–(v), with 'SPF' replaced by 'KPF', plus one additional rule:

   vi. If F is in KPF, then so is the powerset $\mathcal{P}(F)$, defined as $X \mapsto \mathcal{P}(F(X))$ on sets and as $f \mapsto \mathcal{P}(F(f))$ on functions; see (2.13).

   Occasionally, we shall say that a functor F is a *finite* KPF. This means that all the powersets $\mathcal{P}$ occurring in F are actually finite powersets $\mathcal{P}_{\mathrm{fin}}$.

We notice that exponents $(-)^A$ for *finite* sets A are already included in simple polynomial functors via iterated products $F_1 \times \cdots \times F_n$. The collection EPF is typically used to capture coalgebras (or automata) with infinite sets of inputs, given as exponents; see Section 2.2.3 below. The collection KPF is used for non-deterministic computations via powersets; see Section 2.2.4.

The above clauses yield a reasonable collection of functors to start from, but we could of course have included some more constructions in our definition of polynomial functor – such as iterations via initial and final (co)algebras (see Exercise 2.3.8 and e.g. [214, 402, 275]), as used in the experimental programming language Charity [105, 103, 102]. There are thus interesting functors which are out of the 'polynomial scope'; see for instance the multiset or probability distribution functors from Section 4.1 or 'dependent' polynomial functors in Exercise 2.2.6. However, the above clauses suffice for many examples, for the time being.

The coproducts that are used to construct simple polynomial functors are arbitrary, set-indexed coproducts. Frequently we shall use binary versions $F_1 + F_2$, for an index set $I = \{1, 2\}$. But we like to go beyond such finite coproducts, for instance in defining the **list functor** $F^\star$, given as infinite coproduct of products:

$$F^\star = \coprod_{n \in \mathbb{N}} F^n \quad \text{where} \quad F^n = \underbrace{F \times \cdots \times F}_{n \text{ times}}. \tag{2.17}$$

Thus, if F is the identity functor, then $F^\star$ maps a set X to the set of lists:

$$X^\star = 1 + X + (X \times X) + (X \times X \times X) + \cdots.$$

The collection SPF of simple polynomial functors is reasonably stable in the sense that it can be characterised in various ways. Below we give one such alternative characterisation; the other one is formulated later on, in Theorem 4.7.8, in terms of preservation properties. The characterisation below uses 'arities' as commonly used in universal algebra to capture the number of arguments in a primitive function symbol. For instance, addition + has arity 2, and minus − has arity 1. These arities will be used more systematically in Section 6.6 to associate a term calculus with a simple polynomial functor.

Definition 2.2.2 An **arity** is given by a set I and a function $\#\colon I \to \mathbb{N}$. It determines a simple polynomial functor $F_\# \colon \mathbf{Sets} \to \mathbf{Sets}$, namely

$$F_\#(X) \stackrel{\text{def}}{=} \coprod_{i\in I} X^{\#i} = \{(i, \vec{x}) \mid i \in I \text{ and } \vec{x} \in X^{\#i}\}. \tag{2.18}$$

We often call such an $F_\#$ an **arity functor**.

In the style of universal algebra one describes the operations of a group via an index set $I = \{\mathsf{s}, \mathsf{m}, \mathsf{z}\}$, with symbols for sum, minus and zero, with obvious arities $\#(\mathsf{s}) = 2$, $\#(\mathsf{m}) = 1$, $\#(\mathsf{z}) = 0$. The associated functor $F_\#$ sends X to $(X \times X) + X + 1$. In general, these arity functors have a form that clearly is 'polynomial'.

Proposition 2.2.3 *The collections of simple polynomial functors and arity functors are isomorphic.*

Proof By construction an arity functor $F_\# = \coprod_{i\in I}(-)^{\#i}$ is a simple polynomial functor, so we concentrate on the converse. We show that each simple polynomial functor F is an arity functor, with index set $F(1)$, by induction on the structure of F.

- If F is the identity functor $X \mapsto X$ we take $I = F(1) = 1$ and $\# = 1\colon 1 \to \mathbb{N}$. Then $F_\#(X) = \coprod_{i\in 1} X^{\#i} \cong \coprod_{i\in 1} X^1 \cong X = F(X)$.

  If F is a constant functor $X \mapsto A$, then we choose as arity the map $\#\colon A \to \mathbb{N}$, which is constantly 0. Then $F_\#(X) = \coprod_{a\in A} X^{\#a} = \coprod_{a\in A} X^0 \cong \coprod_{a\in A} 1 \cong A = F(X)$.
- If F is a product $X \mapsto F_1(X) \times F_2(X)$, we may assume arities $\#_j\colon I_j \to \mathbb{N}$ for $j \in \{1, 2\}$. We now define $\#\colon I_1 \times I_2 \to \mathbb{N}$ as $\#(i_1, i_2) = \#_1 i_1 + \#_2 i_2$. Then:

$$\begin{aligned}
F_\#(X) &= \coprod_{(i_1,i_2)\in I_1\times I_2} X^{(\#i_1+\#_2 i_2)} \\
&\cong \coprod_{i_1\in I_1} \coprod_{i_2\in I_2} X^{\#_1 i_1} \times X^{\#_2 i_2} \\
&\cong \coprod_{i_1\in I_1} X^{\#_1 i_1} \times \coprod_{i_2\in I_2} X^{\#_2 i_2} \\
&\qquad \text{since } Y \times (-) \text{ preserves coproducts (see also Exercise 2.1.7)} \\
&\overset{\text{(IH)}}{\cong} F_1(X) \times F_2(X) \\
&= F(X).
\end{aligned}$$

- If F is a coproduct $X \mapsto \coprod_{j\in J} F_j(X)$, we may assume arities $\#_j \colon I_j \to \mathbb{N}$ by the induction hypothesis. The cotuple $\# = [\#_j]_{j\in J} \colon \coprod_{j\in j} I_j \to \mathbb{N}$ then does the job:

$$\begin{array}{rcl} F_{\#}(X) & = & \coprod_{(j,i)\in\coprod_{j\in J} I_j} X^{\#(j,i)} \\ & \overset{\text{(IH)}}{\cong} & \coprod_{j\in J} \coprod_{i\in I_j} X^{\#_j(i)} \\ & \cong & \coprod_{j\in J} F_j(X) \\ & = & F(X). \end{array}$$

❑

The arities $\# \colon I \to \mathbb{N}$ that we use here are *single-sorted* arities. They can be used to capture operations of the form $n \to 1$, with n inputs, all of the same sort, and a single output, of this same sort. But multi-sorted (or multi-typed) operations such as $\mathsf{even} \colon \mathbb{N} \to \mathsf{Bool} = \{\mathsf{true}, \mathsf{false}\}$ are out of scope. More generally, given a set of sorts/types S, one can also consider multi-sorted arities as functions $\# \colon I \to S^+ = S^* \times S$. A value $\#(i) = (\langle s_1, \ldots s_n\rangle, t)$ then captures a function symbol with type $s_1 \times \cdots \times s_n \to t$, taking n inputs of sort $s_1, \ldots, s_n$ to an output sort t. Notice that the (single-sorted) arities that we use here are a special case, when the set of sorts S is a singleton $1 = \{0\}$, since $1^\star \cong \mathbb{N}$.

In the remainder of this section we shall see several instances of (simple, exponent and Kripke) polynomial functors. This includes examples of fundamental mathematical structures that arise as coalgebras of such functors.

2.2.1 Statements and Sequences

In Chapter 1 we have used program statements (in Section 1.1) and sequences (in Section 1.2) as motivating examples for the study of coalgebras. We briefly review these examples using the latest terminology and notation.

Recall that statements were introduced as functions acting on a state space S, with different output types depending on whether these statements could hang or terminate abruptly because of an exception. These two representations were written as

$$S \longrightarrow \{\bot\} \cup S, \qquad\qquad S \longrightarrow \{\bot\} \cup S \cup (S \times E).$$

Using the notation from the previous section we now write these as:

$$S \longrightarrow 1 + S, \qquad\qquad S \longrightarrow 1 + S + (S \times E).$$

And so we recognise these statements as coalgebras

$$S \longrightarrow F(S), \qquad\qquad S \longrightarrow G(S)$$

of the simple polynomial functors:

$$F = 1 + \mathrm{id} = \big(X \longmapsto 1 + X\big) \quad \text{and} \quad G = 1 + \mathrm{id} + (\mathrm{id} \times E) = \big(X \longmapsto 1 + X + (X \times E)\big).$$

Thus, these functors determine the kind of computations.

Sequence coalgebras, for a fixed set A, were described in Section 1.2 as functions:

$$S \longrightarrow \{\perp\} \cup (A \times S)$$

i.e. as coalgebras:

$$S \longrightarrow 1 + (A \times S)$$

of the simple polynomial functor $1 + (A \times \mathrm{id})$. This functor was called Seq in Example 1.4.4.5. Again, the functor determines the kind of computations: either fail or produce an element in A together with a successor state. We could change this a bit and drop the fail option. In that case, each state yields an element in A with a successor state. This different kind of computation is captured by a different polynomial functor, namely by $A \times \mathrm{id}$. A coalgebra of this functor is a function

$$S \longrightarrow A \times S,$$

as briefly mentioned in the introduction to Chapter 1 (before Section 1.1). Its behaviour will be an infinite sequence of elements of A: since there is no fail option, these behaviour sequences do not terminate. In the next section we shall see how to formalise this as: infinite sequences $A^{\mathbb{N}}$ form the final coalgebra of this functor $A \times \mathrm{id}$.

2.2.2 Trees

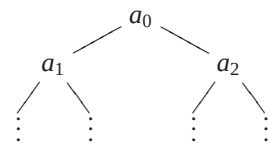
We shall continue this game of capturing different kinds of computation via different polynomial functors. Trees form a good illustration because they occur in various forms. We recall that in computer science trees are usually written upside-down.

Let us start by fixing an arbitrary set A, elements of which will be used as labels in our trees. Binary trees are most common. They arise as outcomes of computations of coalgebras:

$$S \longrightarrow A \times S \times S$$

of the simple polynomial functor $A \times \mathrm{id} \times \mathrm{id}$. Indeed, given a state $x \in S$, a one-step computation yields a triple (a_0, x_1, x_2) of an element $a_0 \in A$ and two

successor states $x_1, x_2 \in S$. Continuing the computation with both x_1 and x_2 yields two more elements in A, and four successor states, etc. This yields for each $x \in S$ an infinite binary tree with one label from A at each node:



In a next step we could consider *ternary* trees as behaviours of coalgebras:

$$S \longrightarrow A \times S \times S \times S$$

of the simple polynomial functor $A \times \mathrm{id} \times \mathrm{id} \times \mathrm{id}$. By a similar extension one can get quaternary trees, etc. These are all instances of finitely branching trees, arising from coalgebras

$$S \longrightarrow A \times S^\star$$

of the Kripke polynomial functor $A \times \mathrm{id}^\star$. Each state $x \in S$ is now mapped to an element in A with a finite sequence $\langle x_1, \ldots x_n \rangle$ of successor states – with which one continues to observe the behaviour of x.

We can ask if the behaviour trees should always be infinitely deep. Finiteness must come from the possibility that a computation fails and yields no successor state. This can be incorporated by considering coalgebras

$$S \longrightarrow 1 + (A \times S \times S)$$

of the simple polynomial functor $1 + (A \times \mathrm{id} \times \mathrm{id})$. Notice that the resulting behaviour trees may be finite in one branch, but infinite in the other. There is nothing in the shape of the polynomial functor that will guarantee that the whole behaviour will actually be finite.

A minor variation is in replacing the set 1 for non-termination by another set B, in

$$S \longrightarrow B + (A \times S \times S).$$

The resulting behaviour trees will have elements from A at their (inner) nodes and from B at the leaves.

2.2.3 Deterministic Automata

Automata are very basic structures in computing, used in various areas: language theory, text processing, complexity theory, parallel and distributed computing, circuit theory, etc. Their state-based, dynamical nature makes

them canonical examples of coalgebras. There are many versions of automata, but here, in this section, we shall concentrate on the two most basic ones: deterministic and non-deterministic. For more information on the extensive theory of automata, see e.g. [42, 127, 416, 30], and on coalgebras and automata see e.g. [412, 457, 319, 196, 429].

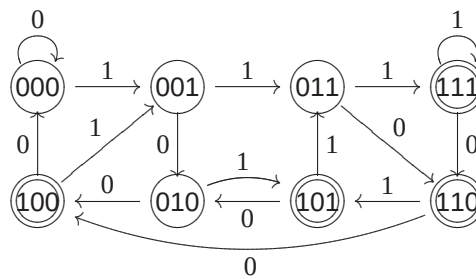
A deterministic automaton is usually defined as consisting of a set S of states, a set A of labels (or actions or letters of an alphabet), a transition function $\delta\colon S \times A \to S$ and a set $F \subseteq S$ of final states. Sometimes, also an initial state $x_0 \in S$ is considered part of the structure, but here it is not. Such an automaton is called deterministic because for each state $x \in S$, and input $a \in A$, there is precisely one successor state $x' = \delta(x, a) \in S$.

First, we shall massage these ingredients into coalgebraic shape. Via the bijective correspondence (2.11) for exponents, the transition function $S \times A \to S$ can also be understood as a map $S \to S^A$. And the subset $F \subseteq S$ of final states corresponds to a characteristic function $S \to \{0, 1\}$. These two functions $S \to S^A$ and $S \to \{0, 1\}$ can be combined into a single function,

$$S \xrightarrow{\langle \delta, \epsilon \rangle} S^A \times \{0, 1\}, \tag{2.19}$$

using the product correspondences (2.2). Thus, deterministic automata with A as a set of labels are coalgebras of the exponent polynomial functor $\mathrm{id}^A \times \{0, 1\}$.

An example of such an automaton with input set $A = \{0, 1\}$ is



(2.20)

The states are the numbers $0, 1, \ldots, 7$ in binary notation, giving as state space:

$$S = \{000, 001, 010, 011, 100, 101, 110, 111\}.$$

Using the standard convention, final states are doubly encircled. In our coalgebraic representation (2.19) this is captured by the function $\epsilon\colon S \to \{0, 1\}$ defined by

$$\epsilon(111) = \epsilon(100) = \epsilon(101) = \epsilon(110) = 1$$
$$\epsilon(000) = \epsilon(001) = \epsilon(011) = \epsilon(010) = 0.$$

The transition function $\delta\colon S \to S^{\{0,1\}}$ is

$$\begin{array}{llll} \delta(000)(0) = 000 & \delta(001)(0) = 010 & \delta(011)(0) = 110 & \delta(111)(0) = 110 \\ \delta(000)(1) = 001 & \delta(001)(1) = 011 & \delta(011)(1) = 111 & \delta(111)(1) = 111 \\ \\ \delta(100)(0) = 000 & \delta(010)(0) = 100 & \delta(101)(0) = 010 & \delta(110)(0) = 111 \\ \delta(100)(1) = 001 & \delta(010)(1) = 101 & \delta(101)(1) = 011 & \delta(110)(1) = 101. \end{array}$$

It is clear that the graphical representation (2.20) is more pleasant to read than these listings. Shortly we discuss what this automaton does.

First we shall stretch the representation (2.19) a little bit and replace the set $\{0, 1\}$ of observable values by an arbitrary set B of outputs. Thus, what shall call a **deterministic automaton** is a coalgebra

$$S \xrightarrow{\langle \delta, \epsilon \rangle} S^A \times B \tag{2.21}$$

of the exponent polynomial functor $\mathrm{id}^A \times B$. We recall that if the set A of inputs is finite, we have a *simple* polynomial functor.

This kind of coalgebra, or deterministic automaton, $\langle \delta, \epsilon \rangle\colon S \to S^A \times B$, thus consists of a **transition function** $\delta\colon S \to S^A$ and an **observation function** $\epsilon\colon S \to B$. For such an automaton, we shall frequently use a transition notation $x \xrightarrow{a} x'$ for $x' = \delta(x)(a)$. Also, we introduce a notation for observation: $x \downarrow b$ stands for $\epsilon(x) = b$. Finally, a combined notation is sometimes useful: $(x \downarrow b) \xrightarrow{a} (x' \downarrow b')$ means three things at the same time: $x \downarrow b$, $x \xrightarrow{a} x'$ and $x' \downarrow b'$.

Often one considers automata with a *finite* state space. This is not natural in a coalgebraic setting, because the state space is considered to be a black box, about which essentially nothing is known – except what can be observed via the operations. Hence, in general, we shall work with arbitrary state spaces, without assuming finiteness. But Section 2.2.6 illustrates how to model n-state systems, when the number of states is a known number n. Additionally, one can consider coalgebras in categories with 'finite' objects. For instance, one can look at coalgebras $S \to S^A \times B$ in the subcategory **FinSets** $\hookrightarrow$ **Sets** of finite sets and (all) functions. This works only when the sets A, B are both finite. More generally, one can work in categories of finitely presented objects [351] or of finitely generated objects [352].

Let $x \in S$ be an arbitrary state of such a coalgebra/deterministic automaton $\langle \delta, \epsilon \rangle\colon S \to S^A \times B$. Applying the output function $\epsilon\colon S \to B$ to x yields an immediate observation $\epsilon(x) \in B$. For each element $a_1 \in A$ we can produce a successor state $\delta(x)(a_1) \in S$; it also gives rise to an immediate observation $\epsilon(\delta(x)(a_1)) \in B$, and for each $a_2 \in A$ a successor state $\delta(\delta(x)(a_1))(a_2) \in S$, etc. Thus, for each finite sequence $\langle a_1, \ldots, a_n \rangle \in A^\star$ we can observe an element $\epsilon(\delta(\cdots\delta(x)(a_1)\cdots)(a_n)) \in B$. Everything we can possibly observe about the

state x is obtained in this way, namely as a function $A^\star \to B$. Such behaviours will form the states of the final coalgebra; see Proposition 2.3.5 in the next section.

For future reference we shall be a bit more precise about lists of inputs. Behaviours can best be described via an iterated transition function

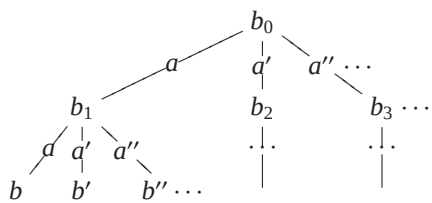
$$S \times A^\star \xrightarrow{\delta^*} S \qquad \text{defined as} \qquad \begin{cases} \delta^*(x, \langle\rangle) = x \\ \delta^*(x, a \cdot \sigma) = \delta^*(\delta(x)(a), \sigma). \end{cases} \tag{2.22}$$

This iterated transition function δ^* gives rise to the multiple-step transition notation: $x \xrightarrow{\sigma}{}^* y$ stands for $y = \delta^*(x, \sigma)$ and means that y is the (non-immediate) successor state of x obtained by applying the inputs from the sequent $\sigma \in A^\star$, from left to right.

The behaviour $\mathrm{beh}(x)\colon A^\star \to B$ of a state $x \in S$ is then obtained as the function that maps a finite sequence $\sigma \in A^\star$ of inputs to the observable output

$$\mathrm{beh}(x)(\sigma) \stackrel{\text{def}}{=} \epsilon(\delta^*(x, \sigma)) \in B. \tag{2.23}$$

An element of the set $B^{A^\star}$ of all such behaviours can be depicted as a rooted tree with elements from the set A of inputs as labels, and elements from the set B of outputs as nodes. For example, a function $\varphi \in B^{A^\star}$ can be described as an infinite tree:



where

$$\begin{aligned} \varphi(\langle\rangle) &= b_0 \\ \varphi(\langle a\rangle) &= b_1 \\ \varphi(\langle a'\rangle) &= b_2 \\ \varphi(\langle a''\rangle) &= b_3 \\ &\text{etc.} \\ \varphi(\langle a, a\rangle) &= b \\ \varphi(\langle a, a'\rangle) &= b' \\ \varphi(\langle a, a''\rangle) &= b'' \\ &\text{etc.} \end{aligned}$$

If the behaviour $\mathrm{beh}(x)$ of a state x looks like this, then one can immediately observe $b_0 = \epsilon(x)$, observe $b_1 = \epsilon(\delta(x)(a))$ after input a, observe $b = \epsilon(\delta(\delta(x)(a))(a)) = \epsilon(\delta^*(x, \langle a, a\rangle))$ after inputting a twice, etc. Thus, there is an edge $b \xrightarrow{a} b'$ in the tree if and only if there are successor states y, y' of x with $(y \downarrow b) \xrightarrow{a} (y' \downarrow b')$. In the next section we shall see (in Proposition 2.3.5) that these behaviours in $B^{A^\star}$ themselves carry the structure of a deterministic automaton.

In the illustration (2.20) one has, for a list of inputs $\sigma \in \{0, 1\}^\star$,

$$\mathrm{beh}(000)(\sigma) = 1 \iff \sigma \text{ contains a 1 in the third position from its end.}$$

In order to see the truth of this statement, notice that the names of the states are chosen in such a way that their bits represent the last three bits that have been consumed so far from the input string σ. In Corollary 2.3.6.2 we shall see more generally that these behaviour maps beh capture the language accepted by the automaton.

Here is a very special way to obtain deterministic automata. A standard result (see e.g. [222, 8.7]) in the theory of differential equations says that unique solutions to such equations give rise to monoid actions; see Exercises 1.4.1 and 2.2.9. Such a monoid action may be of the form $X \times \mathbb{R}_{\geq 0} \to X$, where X is the set of states and $\mathbb{R}_{\geq 0}$ is the set of non-negative reals with monoid structure $(+, 0)$ describing the input (which may be understood as time). In this context monoid actions are sometimes called *flows*, motions, solutions or trajectories; see Exercise 2.2.11 for an example.

Exercise 2.2.12 below gives an account of linear dynamical systems which is very similar to the approach to deterministic automata that we described above. It is based on the categorical analysis by Arbib and Manes [35, 36, 39, 38, 40] of Kalman's [287] module-theoretic approach to linear systems.

Lemma 2.2.4 contains a description of what coalgebra homomorphisms are for automata as in (2.21).

2.2.4 Non-deterministic Automata and Transition Systems

Deterministic automata have a transition *function* $\delta\colon S \times A \to S$. For non-deterministic automata one replaces this function by a *relation*. A state can then have multiple successor states – which is the key aspect of non-determinism. There are several, equivalent ways to represent this. For example, one can replace the transition function $S \times A \to S$ by a function $S \times A \to \mathcal{P}(S)$, yielding for each state $x \in S$ and input $a \in A$ a set of successor states. Of course, this function can also be written as $S \to \mathcal{P}(S)^A$, using currying. This is the same as using a transition relation, commonly written as an arrow: $\to \subseteq S \times A \times S$. Or alternatively, one can use a function $S \to \mathcal{P}(A \times S)$. All this amounts to the same, because of the bijective correspondences from Section 2.1:

$$\dfrac{\dfrac{\dfrac{S \longrightarrow \mathcal{P}(S)^A}{S \times A \longrightarrow \mathcal{P}(S)}\,(2.11)}{\text{relations} \subseteq (S \times A) \times S}\,(2.16)}{\dfrac{\text{relations} \subseteq S \times (A \times S)}{S \longrightarrow \mathcal{P}(A \times S)}\,(2.16)}$$

We have a preference for the first representation and will thus use functions $\delta\colon S \to \mathcal{P}(S)^A$ as non-deterministic transition structures.

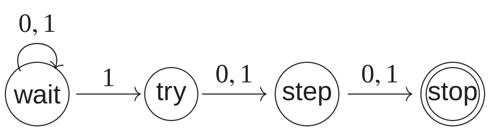
(It should be noted that if we use the finite powerset $\mathcal{P}_{\text{fin}}$ instead of the ordinary one $\mathcal{P}$, then there are no such bijective correspondences, and there is then a real choice of representation.)

Standardly, non-deterministic automata are also considered with a subset $F \subseteq S$ of final states. As for deterministic automata, we like to generalise this subset to an observation function $S \to B$. This leads us to the following definition. A **non-deterministic automaton** is a coalgebra

$$S \xrightarrow{\langle \delta, \epsilon \rangle} \mathcal{P}(S)^A \times B \tag{2.24}$$

of a Kripke polynomial functor $\mathcal{P}(\text{id})^A \times B$, where A is the set of its inputs and B the set of its outputs or observations. As before, the coalgebra consists of a **transition function** $\delta\colon S \to \mathcal{P}(S)^A$ and an **observation function** $\epsilon\colon S \to B$.

As illustration we give a non-deterministic version of the (deterministic) automaton in (2.20) trying to find a 1 in the third position from the end of the input string. As is often the case, the non-deterministic version is much simpler. We use as inputs $A = \{0, 1\}$ and outputs $B = \{0, 1\}$, for observing final states, in



(2.25)

More formally, the state space is $S = \{\mathsf{wait}, \mathsf{try}, \mathsf{step}, \mathsf{stop}\}$ and $\epsilon\colon S \to \{0, 1\}$ outputs 1 only on the state stop. The transition function $\delta\colon S \to \mathcal{P}(S)^{\{0,1\}}$ is

$$\begin{array}{rclcrcl}
\delta(\mathsf{wait})(0) & = & \{\mathsf{wait}\} & \qquad & \delta(\mathsf{try})(0) & = & \{\mathsf{step}\} \\
\delta(\mathsf{wait})(1) & = & \{\mathsf{wait}, \mathsf{try}\} & & \delta(\mathsf{try})(1) & = & \{\mathsf{step}\} \\
\delta(\mathsf{step})(0) & = & \{\mathsf{stop}\} & & \delta(\mathsf{stop})(0) & = & \emptyset \\
\delta(\mathsf{step})(1) & = & \{\mathsf{stop}\} & & \delta(\mathsf{stop})(1) & = & \emptyset.
\end{array}$$

Clearly, we need sets of states to properly describe this automaton coalgebraically.

For non-deterministic automata, as in (2.24), we shall use the same notation as for deterministic ones: $x \xrightarrow{a} x'$ stands for $x' \in \delta(x)(a)$, and $x \downarrow b$ means $\epsilon(x) = b$. New notation is $x \overset{a}{\nrightarrow}$, which means $\delta(x)(a) = \emptyset$; i.e. there is no successor state x' such that x can do an a-step $x \xrightarrow{a} x'$ to x'. In general, for a given x and a there may be multiple (many or no) x' with $x \xrightarrow{a} x'$, but there is precisely one b with $x \downarrow b$. Also in the non-deterministic case we use the combined notation $(x \downarrow b) \xrightarrow{a} (x' \downarrow b')$ for $x \downarrow b$ and $x \xrightarrow{a} x'$ and $x' \downarrow b'$.

As for deterministic automata we wish to define the behaviour of a state from transitions $(x \downarrow b) \xrightarrow{a} (x' \downarrow b')$ by omitting the states and keeping the inputs and outputs. But in the non-deterministic case things are not so easy because there may be multiple successor states. More technically, there are no final coalgebras for functors $\mathcal{P}(\mathrm{id})^A \times B$ describing non-deterministic automata; see Proposition 2.3.8 and the discussion in the preceding paragraph. However, if we restrict ourselves to the *finite* powerset $\mathcal{P}_{\mathrm{fin}}$, final coalgebras do exist. For general non-deterministic automata (2.24), with the proper (non-finite) powerset, one can consider other forms of behaviour, such as traces; see Section 5.3.

Now that we have some idea of what a non-deterministic automaton is, namely a coalgebra $S \to \mathcal{P}(S)^A \times B$, we can introduce various transition systems as special cases.***

- An **unlabelled transition system** (UTS) is a non-deterministic automaton with trivial inputs and outputs: $A = 1$ and $B = 1$. It is thus nothing else but a relation $\to \subseteq S \times S$ on a set of states. UTSs are very basic dynamical systems, but they are important for instance as a basis for model checking: automatic state exploration for proving or disproving properties about systems; see for instance [347, 128].
- A **labelled transition system** (LTS), introduced in [383], is a non-deterministic automaton with trivial output: $B = 1$. It can be identified with a relation $\to \subseteq S \times A \times S$. Labelled transition systems play an important role in the theory of processes, see e.g. [67].
- A **Kripke structure** is a non-deterministic automaton of the form $S \longrightarrow \mathcal{P}(S) \times \mathcal{P}(\mathsf{AtProp})$ with trivial input ($A = 1$) and special output: $B = \mathcal{P}(\mathsf{AtProp})$, for a set AtProp of atomic propositions. The transition function $\delta\colon S \to \mathcal{P}(S)$ thus corresponds to an unlabelled transition system. And the observation function function $\epsilon\colon S \to \mathcal{P}(\mathsf{AtProp})$ tells for each state which of the atomic propositions are true (in that state). Such a function, when written as $\mathsf{AtProp} \to \mathcal{P}(S)$, is often called a valuation. Kripke structures are fundamental in the semantics of modal logic; see e.g. [73] or [129]. The latter reference describes Kripke structures for 'multiple agents', that is, as coalgebras $S \longrightarrow \mathcal{P}(S) \times \cdots \times \mathcal{P}(S) \times \mathcal{P}(\mathsf{AtProp})$ with multiple transition functions.

When we consider (non-)deterministic automata as coalgebras, we get an associated notion of (coalgebra) homomorphism for free. As we shall see next, such a homomorphism both preserves and reflects the transitions. The proofs are easy and are left to the reader.

Lemma 2.2.4 *1. Consider two deterministic automata $X \to X^A \times B$ and $Y \to Y^A \times B$ as coalgebras. A function $f : X \to Y$ is then a homomorphism of coalgebras if and only if*

i. $x \downarrow b \implies f(x) \downarrow b$;
ii. $x \xrightarrow{a} x' \implies f(x) \xrightarrow{a} f(x')$.

2. Similarly, for two non-deterministic automata $X \to \mathcal{P}(X)^A \times B$ and $Y \to \mathcal{P}(Y)^A \times B$ a function $f : X \to Y$ is a homomorphism of coalgebras if and only if

i. $x \downarrow b \implies f(x) \downarrow b$;
ii. $x \xrightarrow{a} x' \implies f(x) \xrightarrow{a} f(x')$;
iii. $f(x) \xrightarrow{a} y \implies \exists x' \in X.\ f(x') = y \wedge x \xrightarrow{a} x'$.

Such a function f is sometimes called a zig-zag morphism; see [63]. ❑

Note that the analogue of point (iii) in (2) trivially holds for deterministic automata. Later, in Chapter 4, we shall see that if we replace the powerset functor $\mathcal{P}$ in the coalgebraic description $X \to \mathcal{P}(X)^A \times B$ of non-deterministic automata by another functor, then we obtain a coalgebraic description of for instance probabilistic or weighted automata.

2.2.5 Context-Free Grammars

A **context-free grammar** (CFG) is a basic formalism in computer science to describe the syntax of programming languages via so-called production rules. These rules are of the form $v \longrightarrow \sigma$, where v is a non-terminal symbol and σ is a finite list of both terminal and non-terminal symbols. If we write V for the set of non-terminals, and A for the terminals, then a CFG is a coalgebra of the form

$$V \xrightarrow{g} \mathcal{P}((V + A)^\star).$$

A CFG is thus a coalgebra of the Kripke polynomial functor $X \longmapsto \mathcal{P}((X+A)^\star)$. It sends each non-terminal v to a set $g(v) \subseteq (V + A)^\star$ of right-hand sides $\sigma \in g(v)$ in productions $v \longrightarrow \sigma$.

A word $\tau \in A^\star$ – with terminals only – can be generated by a CFG g if there is a non-terminal $v \in V$ from which τ arises by applying rules repeatedly. The collection of all such strings is the language that is generated by the grammar.

A simple example is the grammar with $V = \{v\}$, $A = \{a, b\}$ and $g(v) = \{\langle\rangle, avb\}$. It thus involves two productions $v \longrightarrow \langle\rangle$ and $v \longrightarrow a \cdot v \cdot b$. This grammar generates the language of words $a^n b^n$ consisting of a number of as followed by an equal number of bs. Such a grammar is often written in Backus–Naur form (BNF) as $v ::= \langle\rangle \mid avb$.

A derivation of a word imposes a structure on the word that is generated. This structure may be recognised in an arbitrary word in a process called **parsing**. This is one of the very first things a compiler does (after lexical analysis); see for instance [33]. Example 5.3.2 describes the parsed language associated with a CFG via a trace semantics defined by coinduction.

2.2.6 Turing-Style Machines

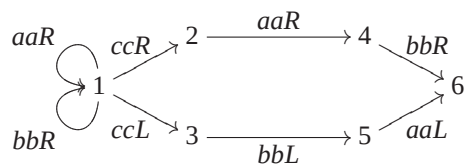
Turing machines are paradigmatic models of computation that are of fundamental importance in the theory of computation (see e.g. [367]), in particular for the concept of computation itself. Turing machines are special automata involving a head moving over a tape that can read and write symbols, following certain rules determined by a finite state machine. Turing machines come in different flavours, with different kinds of tapes or different kinds of computation (e.g. (non-)deterministic or probabilistic, or even of quantum kind). Here we give a sketch of how to describe them coalgebraically, following [249].

The coalgebraic representation requires some care because a Turing machines involves two kinds of states, namely 'register' states given by the contents of the tape and 'steering' states in the finite state machine that controls the head. The register states may be represented as an infinite list $D^{\mathbb{N}}$ of data cells, on which certain operations (reading and writing) may be performed. We abstract from all this and represent it simply as a coalgebra $S \to F(S)$, where S is the set of register states. In case we know that there are n steering states, the resulting Turing-style machine can be represented as a coalgebra:

$$S \longrightarrow \big(F(n \cdot S)\big)^n . \tag{2.26}$$

Recall that $X^n = X \times \cdots \times X$ is the n-fold power and $n \cdot X = X + \cdots + X$ is the n-fold copower. This representation (2.26) gives for each steering state a separate coalgebra $S \to F(n \cdot S)$ telling how the machine moves to successor states (of both register and steering kind).

We consider a concrete example of a non-deterministic Turing machine that accepts all strings $\sigma \in \{a, b, c\}^\star$ where σ contains a c that is preceded or followed by ab. It can be described diagrammatically as



As usual, the label *xyL* means: if you read symbol x at the current location, then write y and move left. Similarly, *xyR* involves a move to the right.

We model such a machine with a tape with symbols from the alphabet $\Sigma = \{a, b, c\}$. The tape stretches in two dimensions, and so we use the integers $\mathbb{Z}$ as index. Thus the type $\mathbb{T}$ of tapes is given by $\mathbb{T} = \Sigma^{\mathbb{Z}} \times \mathbb{Z}$, consisting of pairs (t, p) where $t\colon \mathbb{Z} \to \Sigma = \{a, b, c\}$ is the tape itself and $p \in \mathbb{Z}$ the current position of the head.

Following the description (2.26) we represent the above non-deterministic Turing machine with six states as a coalgebra:

$$\mathbb{T} \longrightarrow \big(\mathcal{P}(6 \cdot \mathbb{T})\big)^6 . \tag{2.27}$$

Below we describe this coalgebra explicitly by enumerating the input cases, where the six coprojections $\kappa_i\colon \mathbb{T} \to 6 \cdot \mathbb{T} = \mathbb{T} + \mathbb{T} + \mathbb{T} + \mathbb{T} + \mathbb{T} + \mathbb{T}$ describe the successor states.

$$(t, p) \longmapsto \begin{cases} 1 \longmapsto \{\kappa_1(t, p+1) \mid t(p) = a \text{ or } t(p) = b\} \cup \\ \qquad\quad \{\kappa_2(t, p+1) \mid t(p) = c\} \cup \{\kappa_3(t, p-1) \mid t(p) = c\} \\ 2 \longmapsto \{\kappa_4(t, p+1) \mid t(p) = a\} \\ 3 \longmapsto \{\kappa_5(t, p-1) \mid t(p) = b\} \\ 4 \longmapsto \{\kappa_6(t, p+1) \mid t(p) = b\} \\ 5 \longmapsto \{\kappa_6(t, p-1) \mid t(p) = a\} \\ 6 \longmapsto \emptyset. \end{cases} \tag{2.28}$$

Notice that we need the powerset $\mathcal{P}$ to capture the non-determinism (especially at state 1).

Later on, in Exercise 5.1.6, we shall see that the functor $X \mapsto (\mathcal{P}(6 \cdot X))^6$ used in (2.27) carries a monad structure that makes it possible to compose this Turing coalgebra with itself, as in Exercise 1.1.2, so that transitions can be iterated.

2.2.7 Non-Well-Founded Sets

Non-well-founded sets form a source of examples of coalgebras which have been of historical importance in the development of the area. Recall that in ordinary set theory there is a foundation axiom (see e.g. [141, chapter II, §5]) stating that there are no infinite descending $\in$-chains $\cdots \in x_2 \in x_1 \in x_0$. This foundation axiom is replaced by an anti-foundation axiom in [140] and in [9], allowing for non-well-founded sets. The second reference [9] received much attention; it formulated an anti-foundation axiom as: every graph has a unique decoration. This can be reformulated easily in coalgebraic terms, stating that

the universe of non-well-founded sets is a final coalgebra for the (special) powerset functor $\wp$ on the category of classes and functions:

$$\wp(X) = \{U \subseteq X \mid U \text{ is a small set}\}.$$

Incidentally, the 'initial algebra' (see Section 2.4) of this functor is the ordinary universe of well-founded sets; see [453, 451] for details.

Aczel developed his non-well-founded set theory in order to provide a semantics for Milner's theory CCS [354] of concurrent processes. An important contribution of this work is the link it established between the proof principle in process theory based on bisimulations (going back to [354, 371]) and coinduction as proof principle in the theory of coalgebras – which will be described in Section 3.4. This work formed a source of much inspiration in the semantics of programming languages [453] and also in logic [60].

Exercises

2.2.1 Check that a polynomial functor which does not contain the identity functor is constant.

2.2.2 Describe the kind of trees that can arise as behaviours of coalgebras:

1. $S \longrightarrow A + (A \times S)$.
2. $S \longrightarrow A + (A \times S) + (A \times S \times S)$.

2.2.3 Check, using Exercise 2.1.10, that non-deterministic automata $X \to \mathcal{P}(X)^A \times 2$ can equivalently be described as transition systems $X \to \mathcal{P}(1 + (A \times X))$. Work out the correspondence in detail.

2.2.4 Describe the arity # for the functors

1. $X \mapsto B + (X \times A \times X)$.
2. $X \mapsto A_0 \times X^{A_1} \times (X \times X)^{A_2}$, for finite sets A_1, A_2.

2.2.5 Check that *finite* arity functors correspond to simple polynomial functors in the construction of which all constant functors $X \mapsto A$ and exponents X^A have finite sets A.

2.2.6 Consider an indexed collection of sets $(A_i)_{i \in I}$ and define the associated 'dependent' polynomial functor **Sets** $\to$ **Sets** by

$$X \longmapsto \coprod_{i \in I} X^{A_i} = \{(i, f) \mid i \in I \wedge f \colon A_i \to X\}.$$

1. Prove that we get a functor in this way; obviously, by Proposition 2.2.3, each polynomial functor is of this form, for a finite set A_i.
2. Check that all simple polynomial functors are dependent – by finding suitable collections $(A_i)_{i \in I}$ for each of them.

(These functors are studied as 'containers' in the context of so-called W-types in dependent type theory for well-founded trees; see for instance [3, 2, 368].)

2.2.7 Recall from (2.13) and (2.14) that the powerset functor $\mathcal{P}$ can be described both as a covariant functor **Sets** $\to$ **Sets** and as a contravariant one $2^{(-)}$: **Sets**$^{\mathrm{op}} \to$ **Sets**. In the definition of Kripke polynomial functors we use the powerset $\mathcal{P}$ covariantly. The functor $\mathcal{N} = \mathcal{P}\mathcal{P}$: **Sets** $\to$ **Sets** is obtained by using the contravariant powerset functor twice – yielding a covariant, but not Kripke polynomial functor. Coalgebras of this so-called neighbourhood functor are used in [423, 358] as models of a special modal logic (see also [194, 193] for the explicitly coalgebraic view).

1. Describe the action $\mathcal{N}(f)\colon \mathcal{N}(X) \to \mathcal{N}(Y)$ of a function $f\colon X \to Y$.
2. Try to see a coalgebra $c\colon X \to \mathcal{N}(X)$ as the setting of a two–player game, with the first player's move in state $x \in X$ given by a choice of a subset $U \in c(x)$ and the second player's reply by a choice of successor state $x' \in U$.

2.2.8 1. Notice that the behaviour function $\mathrm{beh}\colon S \to B^{A^\star}$ from (2.23) for a deterministic automaton satisfies:

$$\begin{aligned} \mathrm{beh}(x)(\langle\rangle) &= \epsilon(x) \\ &= b \qquad\qquad \text{where } x \downarrow b \\ \mathrm{beh}(x)(a \cdot \sigma) &= \mathrm{beh}(\delta(x)(a))(\sigma) \\ &= \mathrm{beh}(x')(\sigma) \qquad \text{where } x \xrightarrow{a} x'. \end{aligned}$$

2. Consider a homomorphism $f\colon X \to Y$ of coalgebras/deterministic automata from $X \to X^A \times B$ and $Y \to Y^A \times B$ and prove that for all $x \in X$,

$$\mathrm{beh}_2(f(x)) = \mathrm{beh}_1(x).$$

2.2.9 Check that the iterated transition function $\delta^*\colon S \times A^\star \to S$ of a deterministic automaton is a monoid action – see Exercise 1.4.1 – for the free monoid structure on $A^\star$ from Exercise 1.4.4.

2.2.10 Note that a function space S^S carries a monoid structure given by composition. Show that the iterated transition function δ^* for a deterministic automaton, considered as a monoid homomorphism $A^\star \to S^S$, is actually obtained from δ by freeness of $A^\star$ – as described in Exercise 1.4.4.

2.2.11 Consider a very simple differential equation of the form $df/dy = -Cf$, where $C \in \mathbb{R}$ is a fixed positive constant. The solution is usually

described as $f(y) = f(0) \cdot e^{-Cy}$. Check that it can be described as a monoid action $\mathbb{R} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$, namely $(x, y) \mapsto xe^{-Cy}$, where $\mathbb{R}_{\geq 0}$ is the monoid of non-negative real numbers with addition $+, 0$.

2.2.12 Let **Vect** be the category with finite-dimensional vector spaces over the real numbers $\mathbb{R}$ (or some other field) as objects, and with linear transformations between them as morphisms. This exercise describes the basics of linear dynamical systems, in analogy with deterministic automata. It does require some basic knowledge of vector spaces.

1. Prove that the product $V \times W$ of (the underlying sets of) two vector spaces V and W is at the same time a product and a coproduct in **Vect** – the same phenomenon as in the category of monoids; see Exercise 2.1.6. Show also that the singleton space 1 is both an initial and a final object. And notice that an element x in a vector space V may be identified with a linear map $\mathbb{R} \to V$.
2. A **linear dynamical system** [287] consists of three vector spaces: S for the state space, A for input and B for output, together with three linear transformations: an input map $G \colon A \to S$, a dynamics $F \colon S \to S$ and an output map $H \colon S \to B$. Note how the first two maps can be combined via cotupling into one transition function $S \times A \to S$, as used for deterministic automata. Because of the possibility of decomposing the transition function in this linear case into two maps $A \to S$ and $S \to S$, these systems are called *decomposable* by Arbib and Manes [35]. (But this transition function $S \times A \to S$ is not bilinear (i.e. linear in each argument separately), so it does not give rise to a map $S \to S^A$ to the vector space S^A of linear transformations from A to S. Hence we do not have a purely coalgebraic description $S \to S^A \times B$ in this linear setting.)
3. For a vector space A, consider, in the notation of [35], the subset of infinite sequences:

$$A^{\S} = \{\alpha \in A^{\mathbb{N}} \mid \text{only finitely many } \alpha(n) \text{ are non-zero}\}.$$

Equip the set $A^{\S}$ with a vector space structure, such that the insertion map $\mathsf{in} \colon A \to A^{\S}$, defined as $\mathsf{in}(a) = (a, 0, 0, 0, \ldots)$, and shift map $\mathsf{sh} \colon A^{\S} \to A^{\S}$, given as $\mathsf{sh}(\alpha) = (0, \alpha(0), \alpha(1), \ldots)$, are linear transformations. (This vector space $A^{\S}$ may be understood as the space of polynomials over A in one variable. It can be defined as the infinite coproduct $\coprod_{n \in \mathbb{N}} A$ of $\mathbb{N}$-copies of A – which is also called a *copower* and written as $\mathbb{N} \cdot A$;

see [344, III, 3]. It is the analogue in **Vect** of the set of finite sequences $B^\star$ for $B \in \mathbf{Sets}$. This will be made precise in Exercise 2.4.8.)

4. Consider a linear dynamical system $A \xrightarrow{G} S \xrightarrow{F} S \xrightarrow{H} B$ as above and show that the analogue of the behaviour $A^\star \to B$ for deterministic automata (see also [37, 6.3]) is the linear map $A^\S \to B$ defined as

$$(a_0, a_2, \ldots, a_n, 0, 0, \ldots) \longmapsto \sum_{i \leq n} HF^i G a_i.$$

This is the standard behaviour formula for linear dynamical systems; see e.g. [287, 38]. (This behaviour map can be understood as starting from the 'default' initial state $0 \in S$. If one wishes to start from an arbitrary initial state $x \in S$, one gets the formula

$$(a_0, a_2, \ldots, a_n, 0, 0, \ldots) \longmapsto HF^{(n+1)}x + \sum_{i \leq n} HF^i G a_i.$$

It is obtained by consecutively modifying the state x with inputs $a_n, a_{n-1}, \ldots, a_0$.)

2.3 Final Coalgebras

In the previous chapter we have seen the special role that is played by the final coalgebra $A^\infty \to 1 + (A \times A^\infty)$ of sequences, both for defining functions into sequences and for reasoning about them – with 'coinduction'. Here we shall define finality in general for coalgebras and investigate this notion more closely – which leads for example to language acceptance by automata; see Corollary 2.3.6.2. This general definition will allow us to use coinductive techniques for arbitrary final coalgebras. The underlying theme is that in final coalgebras there is no difference between states and their behaviours.

In system-theoretic terms, final coalgebras are of interest because they form so-called minimal representations: they are canonical realisations containing all the possible behaviours of a system. It is this idea that we have already tried to suggest in the previous section when discussing various examples of coalgebras of polynomial functors.

Once we have a final coalgebra (for a certain functor), we can map a state from an arbitrary coalgebra (of the same functor) to its behaviour. This induces a useful notion of equivalence between states, namely equality of the associated behaviours. As we shall see later (in Section 3.4), this is bisimilarity.

We shall start in full categorical generality – building on Definition 1.4.5 – but we quickly turn to concrete examples in the category of sets. Examples of final coalgebras in other categories – such as categories of domains or of metric spaces – may be found in many places, such as [143, 144, 145, 433, 5, 453, 133, 132, 218, 86], but also in Section 5.3.

Definition 2.3.1 Let $\mathbb{C}$ be an arbitrary category with an endofunctor $F\colon \mathbb{C} \to \mathbb{C}$. A **final F-coalgebra** is simply a final object in the associated category $\mathbf{CoAlg}(F)$ of F-coalgebras. Thus, it is a coalgebra $\zeta\colon Z \to F(Z)$ such that for any coalgebra $c\colon X \to F(X)$ there is a unique homomorphism $\mathrm{beh}_c\colon X \to Z$ of coalgebras, as in

$$\begin{pmatrix} F(X) \\ \uparrow c \\ X \end{pmatrix} \dashrightarrow^{\mathrm{beh}_c} \begin{pmatrix} F(Z) \\ \uparrow \zeta \\ Z \end{pmatrix} \qquad \text{i.e.} \qquad \begin{array}{ccc} F(X) & \overset{F(\mathrm{beh}_c)}{\dashrightarrow} & F(Z) \\ {\scriptstyle c}\uparrow & & \uparrow{\scriptstyle \zeta} \\ X & \underset{\mathrm{beh}_c}{\dashrightarrow} & Z \end{array}$$

As before, the dashed arrow notation is used for uniqueness. What we call a behaviour map beh_c is sometimes called an *unfold* or a *coreduce* of c.

A common mistake is to read in this definition 'for any *other* coalgebra $c\colon X \to F(X)$' instead of 'for any coalgebra $c\colon X \to F(X)$'. It is important that we can take ζ for this c; the resulting map beh_ζ is then the identity, by uniqueness. This will for instance be used in the proof of Lemma 2.3.3 below.

Recall from Section 1.2 the discussion (after Example 1.2.2) about the two aspects of unique existence of the homomorphism into the final coalgebra, namely (1) existence (used as coinductive/corecursive definition principle) and (2) uniqueness (used as coinductive proof principle). In the next section we shall see that ordinary induction – from a categorical perspective – also involves such a unique existence property. At this level of abstraction there is thus a perfect duality between induction and coinduction.

This unique existence is in fact all we need about final coalgebras. What these coalgebras precisely look like – what their elements are in **Sets** – is usually not relevant. Nevertheless, in order to become more familiar with this topic of final coalgebras, we shall describe several examples concretely. More general theory about the existence of final coalgebras is developed in Section 4.6.

But first we shall look at two general properties of final coalgebras.

Lemma 2.3.2 *A final coalgebra, if it exists, is determined up to isomorphism.*

Proof This is in fact a general property of final objects in a category: if 1 and $1'$ are both a final object in the same category, then there are unique maps $f\colon 1 \to 1'$ and $g\colon 1' \to 1$ by finality. Thus we have two maps $1 \to 1$, namely the composition $g \circ f$ and of course the identity id_1. But then they must be equal by finality of 1. Similarly, by finality of $1'$ we get $f \circ g = \mathrm{id}_{1'}$. Therefore $1 \cong 1'$. ❑

In view of this result we often talk about *the* final coalgebra of a functor, if it exists.

Earlier, in the beginning of Section 1.2 we have seen that the final coalgebra map $A^\infty \to 1 + (A \times A^\infty)$ for sequences is an isomorphism. This turns out to be a general phenomenon, as observed by Lambek [327]: a final F-coalgebra is a fixed point for the functor F. The proof of this result is a nice exercise in diagrammatic reasoning.

Lemma 2.3.3 *A final coalgebra $\zeta\colon Z \to F(Z)$ is necessarily an isomorphism $\zeta\colon Z \xrightarrow{\cong} F(Z)$.*

Proof The first step towards constructing an inverse of $\zeta\colon Z \to F(Z)$ is to apply the functor $F\colon \mathbb{C} \to \mathbb{C}$ to the final coalgebra $\zeta\colon Z \to F(Z)$, which yields again a coalgebra, namely $F(\zeta)\colon F(Z) \to F(F(Z))$. By finality we get a homomorphism $f\colon F(Z) \to Z$ as in

$$\begin{array}{ccc} F(F(Z)) & \overset{F(f)}{\dashrightarrow} & F(Z) \\ {\scriptstyle F(\zeta)}\uparrow & & \uparrow{\scriptstyle \zeta} \\ F(Z) & \underset{f}{\dashrightarrow} & Z \end{array}$$

The aim is to show that this f is the inverse of ζ. We first consider the composite $f \circ \zeta\colon Z \to Z$ and show that it is the identity. We do so by first observing that the identity $Z \to Z$ is the unique homomorphism $\zeta \to \zeta$. Therefore, it suffices to show that $f \circ \zeta$ is also a homomorphism $\zeta \to \zeta$. This follows from an easy diagram chase:

$$\begin{array}{ccccc} F(Z) & \xrightarrow{F(\zeta)} & F(F(Z)) & \xrightarrow{F(f)} & F(Z) \\ {\scriptstyle \zeta}\uparrow & & {\scriptstyle F(\zeta)}\uparrow & & \uparrow{\scriptstyle \zeta} \\ Z & \xrightarrow{\zeta} & F(Z) & \xrightarrow{f} & Z \end{array} \qquad \mathrm{beh}_\zeta = \mathrm{id}_Z$$

The rectangle on the right is the one defining f, and thus commutes by definition. And the one on the left obviously commutes. Therefore the outer rectangle commutes. This says that $f \circ \zeta$ is a homomorphism $\zeta \to \zeta$, and thus allows us to conclude that $f \circ \zeta = \mathrm{id}_Z$.

But now we are done since the reverse equation $\zeta \circ f = \mathrm{id}_{F(Z)}$ follows from a simple computation:

$$\begin{array}{rll} \zeta \circ f &= F(f) \circ F(\zeta) & \text{by definition of } f \\ &= F(f \circ \zeta) & \text{by functoriality of } F \\ &= F(\mathrm{id}_Z) & \text{as we just proved} \\ &= \mathrm{id}_{F(Z)} & \text{because } F \text{ is a functor.} \end{array}$$ ❑

An immediate negative consequence of this fixed point property of final coalgebras is the following. It clearly shows that categories of coalgebras need not have a final object.

Corollary 2.3.4 *The powerset functor* $\mathcal{P}\colon$ **Sets** $\to$ **Sets** *does not have a final coalgebra.*

Proof A standard result of Cantor (proved by so-called diagonalisation; see e.g. [116, theorem 15.10] or [92, theorem 1.10]) says that there cannot be an injection $\mathcal{P}(X) \rightarrowtail X$, for any set X. This excludes a final coalgebra $X \xrightarrow{\cong} \mathcal{P}(X)$. ❑

As we shall see later in this section, the *finite* powerset functor does have a final coalgebra. But first we shall look at some easier examples. The following result, occurring for example in [342, 399, 237], is simple but often useful. It will be used in Section 4.6 to prove more general existence results for final coalgebras.

Proposition 2.3.5 *Fix two sets A and B and consider the polynomial functor* $\mathrm{id}^A \times B$ *whose coalgebras are deterministic automata. The final coalgebra of this functor is given by the set of behaviour functions* $B^{A^\star}$*, with structure*

$$B^{A^\star} \xrightarrow{\quad \zeta = \langle \zeta_1, \zeta_2 \rangle \quad} (B^{A^\star})^A \times B$$

given by

$$\zeta_1(\varphi)(a) = \lambda \sigma \in A^\star.\, \varphi(a \cdot \sigma) \qquad \text{and} \qquad \zeta_2(\varphi) = \varphi(\langle\rangle).$$

Proof We have to show that for an arbitrary coalgebra/deterministic automaton $\langle \delta, \epsilon \rangle\colon X \to X^A \times B$ there is a unique homomorphism of coalgebras $X \to B^{A^\star}$. For this we take of course the behaviour function $\mathrm{beh}\colon X \to B^{A^\star}$ from the previous section, defined in (2.23) by $\mathrm{beh}(x) = \lambda \sigma \in A^\star.\, \epsilon(\delta^*(x, \sigma))$.

We have to prove that it is the unique function making the following diagram commute.

$$\begin{array}{ccc} X^A \times B & \xrightarrow{\text{beh}^{\text{id}_A} \times \text{id}_B} & (B^{A^\star})^A \times B \\ {\scriptstyle \langle \delta, \epsilon \rangle}\uparrow & & \uparrow{\scriptstyle \zeta = \langle \zeta_1, \zeta_2 \rangle} \\ X & \xrightarrow[\text{beh}]{} & B^{A^\star} \end{array}$$

We prove commutation first. It amounts to two points; see Lemma 2.2.4.1.

$$\begin{aligned} (\zeta_1 \circ \text{beh})(x)(a) &= \zeta_1(\text{beh}(x))(a) \\ &= \lambda\sigma.\, \text{beh}(x)(a \cdot \sigma) \\ &= \lambda\sigma.\, \text{beh}(\delta(x)(a))(\sigma) \qquad \text{see Exercise 2.2.8.1} \\ &= \text{beh}(\delta(x)(a)) \\ &= \text{beh}^{\text{id}_A}(\delta(x))(a) \\ &= (\text{beh}^{\text{id}_A} \circ \delta)(x)(a) \\ (\zeta_2 \circ \text{beh})(x) &= \text{beh}(x)(\langle\rangle) \\ &= \epsilon(\delta^*(x, \langle\rangle)) \\ &= \epsilon(x). \end{aligned}$$

Next we have to prove uniqueness. Assume that $f\colon X \to B^{A^\star}$ is also a homomorphism of coalgebras. Then one can show, by induction on $\sigma \in A^\star$, that for all $x \in X$ one has $f(x)(\sigma) = \text{beh}(x)(\sigma)$:

$$\begin{aligned} f(x)(\langle\rangle) &= \zeta_2(\, f(x)) \\ &= \epsilon(x) && \text{since } f \text{ is a homomorphism} \\ &= \text{beh}(x)(\langle\rangle) \\ f(x)(a \cdot \sigma) &= \zeta_1(\, f(x))(a)(\sigma) \\ &= f(\delta(x)(a))(\sigma) && \text{since } f \text{ is a homomorphism} \\ &= \text{beh}(\delta(x)(a))(\sigma) && \text{by induction hypothesis} \\ &= \text{beh}(x)(a \cdot \sigma) && \text{see Exercise 2.2.8.1.} \end{aligned}$$

❑

There are two special cases of this general result that are worth mentioning explicitly.

Corollary 2.3.6 *Consider the above final coalgebra* $B^{A^\star} \xrightarrow{\cong} (B^{A^\star})^A \times B$ *of the deterministic automata functor* $\text{id}^A \times B$.

1. *When* $A = 1$, *so that* $A^\star = \mathbb{N}$, *the resulting functor* $\text{id} \times B$ *captures stream coalgebras* $X \to X \times B$. *Its final coalgebra is the set* $B^{\mathbb{N}}$ *of infinite sequences (streams) of elements of* B, *with* (*tail*, *head*) *structure,*

$$B^{\mathbb{N}} \xrightarrow{\cong} B^{\mathbb{N}} \times B \qquad \textit{given by} \qquad \varphi \longmapsto (\, \lambda n \in \mathbb{N}.\, \varphi(n+1),\ \varphi(0)\,),$$

as described briefly towards the end of the introduction to Chapter 1.

2. *When $B = 2 = \{0, 1\}$ describes final states of the automaton, the final coalgebra $B^{A^\star}$ is the set $\mathcal{P}(A^\star)$ of languages over the alphabet A, with structure*

$$\mathcal{P}(A^\star) \xrightarrow{\quad\cong\quad} \mathcal{P}(A^\star)^A \times \{0, 1\}$$

given by

$$L \longmapsto (\, \lambda a \in A.\, L_a,\ \textit{if } \langle\rangle \in L \textit{ then } 1 \textit{ else } 0\,),$$

where L_a is the so-called a-derivative, introduced by Brzozowski [89], and defined as

$$L_a = \{\sigma \in A^\star \mid a \cdot \sigma \in L\}.$$

*Given an arbitrary automaton $\langle \delta, \epsilon \rangle \colon X \to X^A \times \{0, 1\}$ of this type, the resulting behaviour map $\mathrm{beh}_{\langle\delta,\epsilon\rangle} \colon X \to \mathcal{P}(A^\star)$ thus describes the language $\mathrm{beh}_{\langle\delta,\epsilon\rangle}(x) \subseteq A^\star$ **accepted** by this automaton with $x \in X$ considered as initial state.*

Both these final coalgebras $A^{\mathbb{N}}$ and $\mathcal{P}(A^\star)$ are studied extensively by Rutten; see [410, 412, 413] and Example 3.4.5 later on.

Example 2.3.7 The special case of (1) in the previous result is worth mentioning, where B is the set $\mathbb{R}$ of real numbers (and $A = 1$). We then get a final coalgebra $\mathbb{R}^{\mathbb{N}} \xrightarrow{\cong} (\mathbb{R}^{\mathbb{N}})^{\mathbb{N}} \times \mathbb{R}$ of streams of real numbers. Recall that a function $f \colon \mathbb{R} \to \mathbb{R}$ is called **analytic** if it possesses derivatives of all orders and agrees with its Taylor series in the neighbourhood of every point. Let us write $\mathcal{A}$ for the set of such analytic functions. It carries a coalgebra structure:

$$\begin{array}{ccc} \mathcal{A} & \xrightarrow{\quad d \quad} & \mathcal{A} \times \mathbb{R} \\ f & \longmapsto & (\, f', f(0)). \end{array}$$

Here we write f' for the derivative of f. The induced coalgebra homomorphism $\mathrm{beh}_d \colon \mathcal{A} \to \mathbb{R}^{\mathbb{N}}$ maps an analytic function f to the stream of derivatives at 0, i.e. to $\mathsf{Taylor}(f) = \mathrm{beh}_d(f) = (f(0), f'(0), f''(0), \ldots, f^{(n)}(0), \ldots) \in \mathbb{R}^{\mathbb{N}}$. The output values (in $\mathbb{R}$) in this stream are of course the coefficients in the Taylor series expansion of f in

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n.$$

This shows that beh_d is an isomorphism – and thus that $\mathcal{A}$ can also be considered as the final coalgebra of the functor $(-) \times \mathbb{R}$.

The source of this 'coinductive view on calculus' is [376]. It contains a further elaboration of these ideas. Other coalgebraic 'next' or 'tail' operations

are also studied as derivatives in [412, 413], with the familiar derivative notation $(-)'$ to describe for instance the tail of streams. We may then write $\mathsf{Taylor}(f)' = \mathsf{Taylor}(f')$, so that taking Taylor series commutes with derivatives.

Corollary 2.3.4 implies that there is no final coalgebra for the non-deterministic automata functor $\mathcal{P}(\mathrm{id})^A \times B$. However if we restrict ourselves to the finite powerset functor $\mathcal{P}_{\mathrm{fin}}$ there is a final coalgebra. At this stage we shall be very brief, basically limiting ourselves to the relevant statement. It is a special case of Theorem 2.3.9, the proof of which will be given later, in Section 4.6.

Proposition 2.3.8 (After [55]) *Let A and B be arbitrary sets. Consider the finite Kripke polynomial functor $\mathcal{P}_{fin}(\mathrm{id})^A \times B$ whose coalgebras are image finite non-deterministic automata. This functor has a final coalgebra.*

So far in this section we have seen several examples of final coalgebras. One might wonder which polynomial functors possess a final coalgebra. The following result gives an answer. Its proof will be postponed until later in Section 4.6, because it requires some notions that have not been introduced yet.

Theorem 2.3.9 *Each finite Kripke polynomial functor* **Sets** $\to$ **Sets** *has a final coalgebra.*

As argued before, one does not need to know what a final coalgebra looks like in order to work with it. Its states coincide with its behaviours, so a purely behaviouristic view is justified: unique existence properties are sufficiently strong to use it as a black box. See for instance Exercise 2.3.4 below.

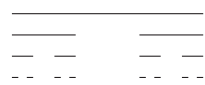
A good question raised explicitly in [411] is: which kind of functions can be defined with coinduction? Put another way: is there, in analogy with the collection of recursively defined functions (on the natural numbers), a reasonable collection of *corecursively* defined functions? This question is still largely open.

2.3.1 Beyond Sets

So far we have concentrated on functors $F\colon$ **Sets** $\to$ **Sets** on the category of sets and functions. Indeed, most of the examples in this book will arise from such functors. It is important however to keep the broader (categorical) perspective in mind and realise that coalgebras are also of interest in other universes. Good examples appear in Section 5.3 where traces of suitable coalgebras are described via coalgebras in the category **SetsRel** of sets with

relations as morphisms. At this stage we shall consider a single example in the category **Sp** of topological spaces and continuous functions between them.

Example 2.3.10 The set $2^{\mathbb{N}} = \mathcal{P}(\mathbb{N})$, for $2 = \{0, 1\}$, of infinite sequences of bits (or subsets of $\mathbb{N}$) carries a topology yielding the well-known 'Cantor space'; see [432, Section 2.3]. Alternatively, this space can be represented as the intersection of a descending chain of intervals (2^{n+1} separate pieces at stage n) of the real interval $[0, 1]$; see below or see or any textbook on topology, for instance [87]:



Starting from the unit interval $[0, 1]$ one keeps in step one the left and right thirds, given by the closed subintervals $[0, \frac{1}{3}]$ and $[\frac{2}{3}, 1]$. These can be described as 'left' and 'right', or as '0' and '1'. In a next step one again keeps the left and right thirds, giving us four closed intervals $[0, \frac{1}{9}]$, $[\frac{2}{9}, \frac{1}{3}]$, $[\frac{2}{3}, \frac{7}{9}]$ and $[\frac{8}{9}, 1]$; they can be referred to via the two-bit words 00, 01, 10, 11, respectively. And so on. The Cantor set is then defined as the (countable) intersection of all these intervals. It is a closed subspace of $[0, 1]$, obtained as an intersection of closed subspaces. It is also totally disconnected.

Elements of the Cantor set can be identified with infinite streams $2^{\mathbb{N}}$, seen as consecutive 'left' or 'right' choices. The basic opens of $2^{\mathbb{N}}$ are the subsets $\uparrow \sigma = \{\sigma \cdot \tau \mid \tau \in 2^{\mathbb{N}}\}$ of infinite sequences starting with σ, for $\sigma \in 2^\star$ a finite sequence.

Recall that **Sp** is the category of topological spaces with continuous maps between them. This category has coproducts, given as in **Sets**, with topology induced by the coprojections. In particular, for an arbitrary topological space X the coproduct $X + X$ carries a topology in which subsets $U \subseteq X + X$ are open if and only if both $\kappa_1^{-1}(U)$ and $\kappa_2^{-1}(U)$ are open in X. The Cantor space can then be characterised as the final coalgebra of the endofunctor $X \mapsto X + X$ on **Sp**.

A brief argument goes as follows. Corollary 2.3.6.1 tells that the set of streams $2^{\mathbb{N}}$ is the final coalgebra of the endofunctor $X \mapsto X \times 2$ on **Sets**. There is an obvious isomorphism $X \times 2 \cong X + X$ of sets (see Exercise 2.1.7), which is actually also an isomorphism of topological spaces if one considers the set 2 with the discrete topology (in which every subset is open); see Exercise 2.3.7 for more details.

But one can of course also check the finality property explicitly in the category **Sp** of topological spaces. The final coalgebra $\zeta \colon 2^{\mathbb{N}} \xrightarrow{\cong} 2^{\mathbb{N}} + 2^{\mathbb{N}}$ is given on $\sigma \in 2^{\mathbb{N}}$ by

$$\zeta(\sigma) = \begin{cases} \kappa_1 \mathsf{tail}(\sigma) & \text{if } \mathsf{head}(\sigma) = 0 \\ \kappa_2 \mathsf{tail}(\sigma) & \text{if } \mathsf{head}(\sigma) = 1. \end{cases}$$

It is not hard to see that both ζ and ζ^{-1} are continuous. For an arbitrary coalgebra $c\colon X \to X + X$ in **Sp** we can describe the unique homomorphism of coalgebra $\mathrm{beh}_c\colon X \to 2^{\mathbb{N}}$ on $x \in X$ and $n \in \mathbb{N}$ as

$$\mathrm{beh}_c(x)(n) = \begin{cases} 0 & \text{if } \exists y.\, c(([\mathrm{id}, \mathrm{id}] \circ c)^n(x)) = \kappa_1 y \\ 1 & \text{if } \exists y.\, c(([\mathrm{id}, \mathrm{id}] \circ c)^n(x)) = \kappa_2 y. \end{cases}$$

Again, this map is continuous. More examples and theory about coalgebras and such fractal-like structure may be found in [331] and [205].

Exercises

2.3.1 Check that a final coalgebra of a monotone endofunction $f\colon X \to X$ on a poset X, considered as a functor, is nothing but a greatest fixed point. (See also Exercise 1.3.5.)

2.3.2 For arbitrary sets A, B, consider the (simple polynomial) functor $X \mapsto (X \times B)^A$. Coalgebras $X \to (X \times B)^A$ of this functor are often called Mealy machines.

1. Check that Mealy machines can equivalently be described as deterministic automata $X \to X^A \times B^A$ and that the final Mealy machine is B^{A^+}, by Proposition 2.3.5, where $A^+ \hookrightarrow A^\star$ is the subset of non-empty finite sequences. Describe the final coalgebra structure $B^{A^+} \to (B^{A^+} \times B)^A$ explicitly.
2. Fix a set A and consider the final coalgebra $A^{\mathbb{N}}$ of the stream functor $A \times (-)$. Define by finality an 'alternation' function $\mathsf{alt}\colon A^{\mathbb{N}} \times A^{\mathbb{N}} \to A^{\mathbb{N}}$, such that

$$\mathsf{alt}(a_0 a_1 a_2 \cdots,\ b_0 b_1 b_2 \cdots) = a_0 b_1 a_2 b_3 \cdots$$

Prove by coinduction

$$\mathsf{alt}(\sigma, \mathsf{alt}(\tau_1, \rho)) = \mathsf{alt}(\sigma, \mathsf{alt}(\tau_2, \rho)).$$

Thus, in such a combination the middle argument is irrelevant.
3. Consider the set Z of so-called causal stream functions, given by

$$Z = \Big\{ \psi\colon A^{\mathbb{N}} \to B^{\mathbb{N}} \;\Big|\; \forall n \in \mathbb{N}.\, \forall \alpha, \alpha' \in A^{\mathbb{N}}.\ (\forall i \leq n.\, \alpha(i) = \alpha'(i)) \Rightarrow \psi(\alpha)(n) = \psi(\alpha')(n) \Big\}.$$

For such a causal stream function ψ, the output $\psi(\alpha)(n) \in B$ is thus determined by the first $n + 1$ elements $\alpha(0), \ldots, \alpha(n) \in A$.

Prove that Z yields an alternative description of the final Mealy automaton, via the structure map $\zeta\colon Z \to (Z \times B)^A$ given by

$$\zeta(\psi)(a) = (\lambda\alpha \in A^{\mathbb{N}}.\, \lambda n.\, \psi(a \cdot \alpha)(n+1),\ \psi(\lambda n \in \mathbb{N}.\, a)(0))$$

where $a \cdot \alpha$ is prefixing to $\alpha \in A^{\mathbb{N}}$, considered as infinite sequence.

For more information on Mealy machines, see [84, 195].

2.3.3 Assume a category $\mathbb{C}$ with a final object $1 \in \mathbb{C}$. Call a functor $F\colon \mathbb{C} \to \mathbb{C}$ affine if it preserves the final object: the map $F(1) \to 1$ is an isomorphism. Prove that the inverse of this map is the final F-coalgebra, i.e. the final object in the category $\mathbf{CoAlg}(F)$. (Only a few of the functors F that we consider are affine; examples are the identity functor the non-empty powerset functor, or the distribution functor $\mathcal{D}$ from Section 4.1. Affine functions occur especially in probabilistic computation.)

2.3.4 Let Z be the (state space of the) final coalgebra of the binary tree functor $X \mapsto 1 + (A \times X \times X)$. Define by coinduction a mirror function $\mathsf{mir}\colon Z \to Z$ which (deeply) exchanges the subtrees. Prove, again by coinduction, that $\mathsf{mir} \circ \mathsf{mir} = \mathrm{id}_Z$. Can you tell what the elements of Z are?

2.3.5 Recall the decimal representation coalgebra $\mathsf{nextdec}\colon [0,1) \to 1 + (\{0,1,\ldots,9\} \times [0,1))$ from Example 1.2.2, with its behaviour map $\mathrm{beh}_{\mathsf{nextdec}}\colon [0,1) \to \{0,1,\ldots,9\}^\infty$. Prove that this behaviour map is a split mono: there is a map e in the reverse direction for which we have $e \circ \mathrm{beh}_{\mathsf{nextdec}} = \mathrm{id}_{[0,1)}$. (The behaviour map is not an isomorphism, because both numbers 5 and $49999\cdots$, considered as sequences in $\{0,1,\ldots,9\}^\infty$, represent the fraction $\frac{1}{2} \in [0,1)$. See other representations as continued fractions in [378] or [366] which do yield isomorphisms.)

2.3.6 This exercise is based on [237, lemma 5.4].

1. Fix three sets A, B, C, and consider the simple polynomial functor

$$X \longmapsto (C + (X \times B))^A.$$

Show that its final coalgebra can be described as the set of functions:

$$Z = \{\varphi \in (C+B)^{A^+} \mid \forall\sigma \in A^+.\, \forall c \in C.\, \varphi(\sigma) = \kappa_1(c) \Rightarrow \forall\tau \in A^\star.\, \varphi(\sigma \cdot \tau) = \kappa_1(c)\}.$$

Once such functions $\varphi \in Z$ hit C, they keep this value in C. Here we write A^+ for the subset of $A^\star$ of non-empty finite sequences.

The associated coalgebra structure $\zeta\colon Z \xrightarrow{\cong} (C + (Z \times B))^A$ is given by

$$\zeta(\varphi)(a) = \begin{cases} \kappa_1(c) & \text{if } \varphi(\langle a\rangle) = \kappa_1(c) \\ \kappa_2(b, \varphi') & \text{if } \varphi(\langle a\rangle) = \kappa_2(b) \text{ where } \varphi'(\sigma) = \varphi(a \cdot \sigma). \end{cases}$$

2. Check that the fact that the set B^∞ of both finite and infinite sequences is the final coalgebra of the functor $X \mapsto 1 + (X \times B)$ is a special case of this.
3. Generalise the result in (1) to functors of the form

$$X \longmapsto (C_1 + (X \times B_1))^{A_1} \times \cdots \times (C_n + (X \times B_n))^{A_n}$$

using this time as state space of the final coalgebra the set

$$\Big\{ \varphi \in (C + B)^{A^+} \;\Big|\; \forall \sigma \in A^\star.\, \forall i \leq n.\; \forall a \in A_i.\, \varphi(\sigma \cdot \kappa_i(a)) \in \kappa_1[\kappa_i[C_i]] \vee \varphi(\sigma \cdot \kappa_i(a)) \in \kappa_2[\kappa_i[B_i]] \wedge \forall c \in C_i.\, \sigma \neq \langle\rangle \wedge \varphi(\sigma) = \kappa_1(\kappa_i(c)) \Rightarrow \forall \tau \in A^\star.\, \varphi(\sigma \cdot \tau) = \kappa_1(\kappa_i(c)) \Big\}$$

where $A = A_1 + \cdots + A_n$, $B = B_1 + \cdots + B_n$ and $C = C_1 + \cdots + C_n$.
4. Show how classes as in (1.10) fit into this last result. *Hint*: Use that $S + (S \times E) \cong (S \times 1) + (S \times E) \cong S \times (1 + E)$, using distributivity from Exercise 2.1.7.

2.3.7 For a topological space A consider the set $A^{\mathbb{N}}$ of streams with the product topology (the least topology that makes the projections $\pi_n\colon A^{\mathbb{N}} \to A$ continuous).

1. Check that the head $A^{\mathbb{N}} \to A$ and tail $A^{\mathbb{N}} \to A^{\mathbb{N}}$ operations are continuous.
2. Prove that the functor $A \times (-)\colon \mathbf{Sp} \to \mathbf{Sp}$ has $A^{\mathbb{N}}$ as final coalgebra.
3. Show that in the special case where A carries the discrete topology (in which every subset is open) the product topology on $A^{\mathbb{N}}$ is given by basic open sets $\uparrow\sigma = \{\sigma \cdot \tau \mid \tau \in A^{\mathbb{N}}\}$, for $\sigma \in A^\star$ as in Example 2.3.10.

2.3.8
1. Note that the assignment $A \mapsto A^{\mathbb{N}}$ yields a functor $\mathbf{Sets} \to \mathbf{Sets}$.
2. Prove the general result: consider a category $\mathbb{C}$ with a functor $F\colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ in two variables. Assume that for each object $A \in \mathbb{C}$, the functor $F(A, -)\colon \mathbb{C} \to \mathbb{C}$ has a final coalgebra $Z_A \xrightarrow{\cong} F(A, Z_A)$. Prove that the mapping $A \mapsto Z_A$ extends to a functor $\mathbb{C} \to \mathbb{C}$.

2.4 Algebras

So far we have talked much about coalgebras. One way to introduce coalgebras is as duals of algebras. We shall do this the other way around and introduce algebras (in categorical formulation) as duals of coalgebras. This reflects of course the emphasis in this book.

There are many similarities (or dualities) between algebras and coalgebras which are often useful as guiding principles. But one should keep in mind that there are also significant differences between algebra and coalgebra. For example, in a computer science setting, algebra is mainly of interest for dealing with *finite* data elements – such as finite lists or trees – using induction as main definition and proof principle. A key feature of coalgebra is that it deals with potentially infinite data elements and with appropriate state-based notions and techniques for handling these objects. Thus, algebra is about construction, whereas coalgebra is about deconstruction – understood as observation and modification.

This section will introduce the categorical definition of algebras for a functor, in analogy with coalgebras of a functor in Definition 1.4.5. It will briefly discuss initiality for algebras, as dual of finality, and will illustrate that it amounts to ordinary induction. Also, it will mention several possible ways of combining algebras and coalgebras. A systematic approach to such combinations using distributive laws will appear in Chapter 5.

We start with the abstract categorical definition of algebras, which is completely dual (i.e. with reversed arrows) to what we have seen for coalgebras.

Definition 2.4.1 Let $\mathbb{C}$ be an arbitrary category, with an endofunctor $F\colon \mathbb{C} \to \mathbb{C}$.

1. An **F-algebra**, or just an **algebra** for F, consists of a 'carrier' object $X \in \mathbb{C}$ together with a morphism $a\colon F(X) \to X$, often called the constructor, or operation.
2. A **homomorphism of algebras**, or a **map of algebras** or **algebra map**, from one algebra $a\colon F(X) \to X$ to another coalgebra $b\colon F(Y) \to Y$ consists of a morphism $f\colon X \to Y$ in $\mathbb{C}$ which preserves the operations:

$$\begin{array}{ccc} F(X) & \xrightarrow{F(f)} & F(Y) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ X & \xrightarrow[f]{} & Y \end{array}$$

This yields a category, for which we shall write **Alg**(F).

3. An **initial** F-algebra is an initial object in the category $\mathbf{Alg}(F)$: it is an algebra $\alpha\colon F(A) \to A$ such that for any F-algebra $b\colon F(X) \to X$ there is a unique homomorphism of algebras int_b from α to b in

$$\begin{pmatrix} F(A) \\ \downarrow{\scriptstyle \alpha} \\ A \end{pmatrix} \dashrightarrow^{\mathrm{int}_b} \begin{pmatrix} F(B) \\ \downarrow{\scriptstyle b} \\ B \end{pmatrix} \qquad \text{i.e.} \qquad \begin{array}{ccc} F(A) & \overset{F(\mathrm{int}_b)}{\dashrightarrow} & F(X) \\ {\scriptstyle \alpha}\downarrow & & \downarrow{\scriptstyle b} \\ X & \underset{\mathrm{int}_b}{\dashrightarrow} & X \end{array}$$

We call this map int_b an interpretation map for the algebra b; sometimes it is also called a *fold* or a *reduce* of b.

Certain maps can be seen both as algebra and as coalgebra. For example, a map $S \times A \to S$ is an algebra – of the functor $X \mapsto X \times A$ – but can also be regarded as a coalgebra $S \to S^A$, after currying (2.11). But for other maps there is more clarity: maps $A \to S$ are algebras, which can be used to construct elements in S from elements in A. And maps $S \to A$ are coalgebras, which provide observations in A about elements in S.

The functor F in the above definition corresponds to what is traditionally called a signature. In simple form a (single-sorted) signature consists of a number of operations op_i, say for $1 \leq i \leq n$, each with their own arity $k_i \in \mathbb{N}$. An algebra for such a signature is then a set S with interpretations $[\![\, \mathsf{op}_i \,]\!]\colon S^{k_i} \to S$. Using the coproduct correspondence (2.6), they may be combined to a single cotuple operation,

$$S^{k_1} + \cdots + S^{k_n} \xrightarrow{[\, [\![\, \mathsf{op}_i \,]\!], \ldots, [\![\, \mathsf{op}_i \,]\!] \,]} S, \tag{2.29}$$

forming an algebra of the simple polynomial functor $X \mapsto X^{k_1} + \cdots + X^{k_n}$. This functor thus captures the number and types of the operations – that is, the signature. In fact, we have already seen the more general description of signatures via arities $\#\colon I \to \mathbb{N}$ and the associated simple polynomial functor $F_\# = \coprod_{i \in I} (-)^{\#i}$, namely in Definition 2.2.2.

A rule of thumb is: data types are algebras, and state-based systems are coalgebras. But this does not always give a clear-cut distinction. For instance, is a stack a data type or does it have a state? In many cases, however, this rule of thumb works: natural numbers are algebras (as we are about to see), and machines are coalgebras. Indeed, the latter have a state that can be observed and modified.

Initial algebras are special, just like final coalgebras. Initial algebras (in **Sets**) can be built as so-called term models: they contain everything that can be built from the operations themselves, and nothing more. Similarly, we saw

that final coalgebras consist of observations only. The importance of initial algebras in computer science was first emphasised in [165]. For example, if F is the signature functor for some programming language, the initial algebra $F(P) \to P$ may be considered as the set of programs, and arbitrary algebras $F(D) \to D$ may be seen as denotational models. Indeed, the resulting unique homomorphism $\mathrm{int} = [\![-]\!] \colon P \to D$ is the semantical interpretation function. It is *compositional* by construction, because it commutes with the operations of the programming language.

Example 2.4.2 In mathematics many algebraic structures are single-sorted (or single-typed). Examples are monoids, groups, rings, etc. In these structures there is only one carrier set involved. Some structures, such as vector spaces, involve two sorts, namely scalars and vectors. In this case one speaks of a multisorted or typed algebra. Still, in mathematics vector spaces are often described as single-sorted, by keeping the field involved fixed.

In computer science most of the examples are multi-sorted. For instance, a specification of some data structure, say a queue, involves several sorts (or types): the type D of data on the queue, the type Q of the queue itself, and possibly some other types like $\mathbb{N}$ for the length of the queue. Type theory has developed into an area of its own, studying various calculi for types and terms.

Most programming languages are typed, involving various types like `nat`, `int`, `float`, `bool`, and possibly also additional user-definable types. We shall describe a toy typed programming language and show how its signature can be described via a functor. Interestingly, this will not be a functor on the standard category **Sets**, but a functor on the product category $\mathbf{Sets}^3 = \mathbf{Sets} \times \mathbf{Sets} \times \mathbf{Sets}$. The 3-fold product $\mathbf{Sets}^3$ is used because our toy language contains only three types, namely N for numbers, B for booleans (true and false) and S for program statements.

We assume that our programming language involves the following operations on numbers and booleans:

$\mathtt{n}\colon$	N	for $0 \leq \mathtt{n} < 2^{16}$
$\mathtt{v}\colon$	N	for variables $\mathtt{v} \in V$
$+\colon$	$N \times N \to N$	for addition
$*\colon$	$N \times N \to N$	for multiplication

$\mathtt{true}, \mathtt{false}\colon$	B	truth values
$\wedge\colon$	$B \times B \to B$	for conjunction
$\neg\colon$	$B \to B$	for negation
$=\colon$	$N \times N \to B$	for equality
$\leq\colon$	$N \times N \to B$	for comparison.

Here we use a (fixed) set V of variables. Note that the two operations $=$ and $\leq$ involve both types N and B. Also statements involve multiple types:

`skip`:	S	the program that does nothing
`;`:	$S \times S \to S$	sequential program composition
`-` `:=` `-`:	$V \times N \to S$	for assignment
`if` – `then` – `else` – `fi`:	$B \times S \times S \to S$	for the conditional statement
`while` – `do` – `od`:	$B \times S \to S$	for repetition.

Following the (single-sorted) signature description as in (2.29) we could capture the constants n, sum $+$ and product $*$ via a functor $X \mapsto 2^{16} + (X \times X) + (X \times X)$. But we need to capture all operations at the same time. Thus we use a functor $F\colon \mathbf{Sets}^3 \to \mathbf{Sets}^3$ in three variables, where the first one, written as X, will be used for the type of naturals, the second one Y for booleans and the third one Z for program statements. The functor F is then given by the following expression:

$$\begin{aligned} F\big(X,Y,Z\big) = \big(& 2^{16} + V + (X \times X) + (X \times X)\,, \\ & (X \times X) + (X \times X) + 2 + (Y \times Y) + Y\,, \\ & 1 + (Z \times Z) + (V \times X) + (Y \times Z \times Z) + (Y \times Z)\big). \end{aligned}$$

An algebra $F(A,B,C) \to (A,B,C)$ in $\mathbf{Sets}^3$ for this functor is given by a triple of sets $(A,B,C) \in \mathbf{Sets}^3$ interpreting naturals, booleans and statements, and a triple of functions $(a,b,c)\colon F(A,B,C) \to (A,B,C)$ giving interpretations of the operations. For instance the function b in the middle is given by a 5-cotuple of the form

$$(A \times A) + (A \times A) + 2 + (B \times B) + B \xrightarrow{b = [b_1, b_2, b_3, b_4, b_5]} B$$

where $b_3\colon 2 \to B$ interprets the two booleans (since $2 = 1 + 1 = \{0,1\}$) and $b_5\colon B \to B$ interprets negation; similarly for the maps a, c in this algebra. We see that signature functors for specific languages can become quite complicated. But at an abstract level we work with arbitrary (polynomial) functors, and such complexity remains hidden.

The initial F-algebra is given by three sets containing all syntactically constructed expressions for naturals, booleans and statements, using the above operations. This will not be proven, but the examples below illustrate how initial algebras consist of terms.

In this context of program language semantics, a final coalgebra is also understood as a canonical operational model of behaviours. The unique homomorphism to the final coalgebra may be seen as an operational interpretation function which is *fully abstract*, in the sense that objects have the same interpretation if and only if they are observationally indistinguishable. This relies on Theorem 3.4.1; see [451–453] for more information.

Compositionality for such models in final coalgebras is an important issue; see Section 5.5.

Because of the duality in the definitions of algebras and coalgebras, certain results can be dualised. For example, Lambek's fixed point result for final coalgebras (Lemma 2.3.3) also holds for initial algebras. The proof is the same, but with arrows reversed.

Lemma 2.4.3 *An initial algebra $F(A) \to A$ of a functor $F\colon \mathbb{C} \to \mathbb{C}$ is an isomorphism $F(A) \xrightarrow{\cong} A$ in $\mathbb{C}$.* ❑

The unique existence in the definition of initiality (again) has two aspects, namely existence corresponding to recursion, or definition by induction, and uniqueness, corresponding to proof by induction. This will be illustrated in two examples.

The natural numbers $\mathbb{N}$ form a trivial but important example of an initial algebra. We shall consider it in some detail, relating the familiar description of induction to the categorical one based on initiality.

Example 2.4.4 (Natural numbers) According to Peano, the most basic operations on the natural numbers $\mathbb{N}$ are zero $0 \in \mathbb{N}$ and successor $S\colon \mathbb{N} \to \mathbb{N}$. Using that an element of $\mathbb{N}$ can be described as an arrow $1 \to \mathbb{N}$, together with the coproduct correspondence (2.6), these two maps can be combined into an algebra

$$1 + \mathbb{N} \xrightarrow{[0,S]} \mathbb{N}$$

of the simple polynomial function $F(X) = 1 + X$. This functor adds a new point $* \in 1$ to an arbitrary set X. It is easy to see that the algebra $[0, S]$ is an isomorphism, because each natural number is either zero or a successor. The isomorphism $[0, S]\colon 1 + \mathbb{N} \xrightarrow{\cong} \mathbb{N}$ is in fact the initial algebra of the functor $F(X) = 1 + X$. Indeed, for an arbitrary F-algebra $[a, g]\colon 1 + X \to X$ with carrier X, consisting of functions $a\colon 1 \to X$ and $g\colon X \to X$, there is a unique homomorphism of algebras $f = \mathrm{int}_{[a,g]}\colon \mathbb{N} \to X$ with

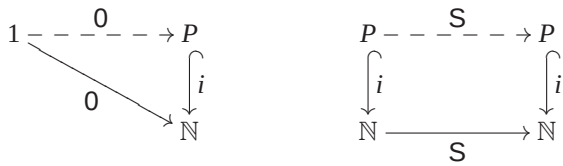
$$\begin{array}{ccc} 1 + \mathbb{N} & \xrightarrow{\mathrm{id}_1 + f} & 1 + X \\ {\scriptstyle [0,S]}\downarrow \cong & & \downarrow{\scriptstyle [a,g]} \\ \mathbb{N} & \xrightarrow[f]{} & X \end{array} \qquad \text{i.e. with} \qquad \begin{cases} f \circ 0 = a \\ f \circ S = g \circ f. \end{cases}$$

In ordinary mathematical language, this says that f is the unique function with $f(0) = a$, and $f(n + 1) = g(f(n))$. This indeed determines f completely, by induction. Thus, ordinary natural number induction implies initiality. We shall illustrate the reverse implication.

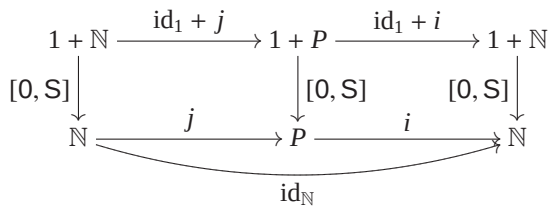
The initiality property of $[0, \mathsf{S}]\colon 1 + \mathbb{N} \xrightarrow{\cong} \mathbb{N}$ is strong enough to prove all of Peano's axioms. This was first shown in [142] (see also [345] or [166]), where the initiality is formulated as 'natural numbers object'. As an example, we shall consider the familiar induction rule: for a predicate $P \subseteq \mathbb{N}$,

$$\frac{P(0) \qquad \forall n \in \mathbb{N}.\, P(n) \implies P(n+1)}{\forall n \in \mathbb{N}.\, P(n)}.$$

In order to show how the validity of this induction rule follows from initiality, let us write i for the inclusion function $P \hookrightarrow \mathbb{N}$. We then note that the assumptions of the induction rule state that the zero and successor functions restrict to P, as in



Thus, the subset P itself carries an algebra structure $[0, \mathsf{S}]\colon 1 + P \to P$. Therefore, by initiality of $\mathbb{N}$ we get a unique homomorphism $j\colon \mathbb{N} \to P$. Then we can show $i \circ j = \mathrm{id}_{\mathbb{N}}$, by uniqueness:



The rectangle on the left commutes by definition of j, and the one on the right by the previous two diagrams. The fact that $i \circ j = \mathrm{id}_{\mathbb{N}}$ now yields $P(n)$, for all $n \in \mathbb{N}$.

Example 2.4.5 (Binary trees) Fix a set A of labels. A signature for A-labelled binary trees may be given with two operations:

nil	for the empty tree,
$\mathsf{node}(b_1, a, b_2)$	for the tree constructed from subtrees b_1, b_2 and label $a \in A$.

Thus, the associated signature functor is $T(X) = 1 + (X \times A \times X)$. The initial algebra will be written as $\mathsf{BinTree}(A)$, with operation

$$1 + (\mathsf{BinTree}(A) \times A \times \mathsf{BinTree}(A)) \xrightarrow[\cong]{[\mathsf{nil}, \mathsf{node}]} \mathsf{BinTree}(A).$$

This carrier set $\mathsf{BinTree}(A)$ can be obtained by considering all terms that can be formed from the constructor nil via repeated application of the node operation $\mathsf{node}(\mathsf{nil}, a, \mathsf{nil})$, $\mathsf{node}(\mathsf{nil}, a', \mathsf{node}(\mathsf{nil}, a, \mathsf{nil}))$, etc. We do not describe it in too much detail because we wish to use it abstractly. Our aim is to traverse such binary trees and collect their labels in a list. We consider two obvious ways to do this – commonly called *inorder* traversal and *preorder* traversal – resulting in two functions $\mathsf{iotrv}, \mathsf{potrv} \colon \mathsf{BinTree}(A) \to A^\star$. These functions can be defined inductively as

$$\begin{aligned} \mathsf{iotrv}(\mathsf{nil}) &= \langle\rangle \\ \mathsf{iotrv}(\mathsf{node}(b_1, a, b_2)) &= \mathsf{iotrv}(b_1) \cdot a \cdot \mathsf{iotrv}(b_2) \\ \mathsf{potrv}(\mathsf{nil}) &= \langle\rangle \\ \mathsf{potrv}(\mathsf{node}(b_1, a, b_2)) &= a \cdot \mathsf{potrv}(b_1) \cdot \mathsf{potrv}(b_2). \end{aligned}$$

They can be defined formally via initiality, by putting two different T-algebra structures on the set $A^\star$ of sequences: the inorder transversal function arises in

$$\begin{array}{ccc} 1+(\mathsf{BinTree}(A) \times A \times \mathsf{BinTree}(A)) & \xrightarrow{\mathrm{id}_1 + (\mathsf{iotrv} \times \mathrm{id}_A \times \mathsf{iotrv})} & 1+(A^\star \times A \times A^\star) \\ \downarrow{\scriptstyle [\mathsf{nil}, \mathsf{node}]} & & \downarrow{\scriptstyle [\langle\rangle, g]} \\ \mathsf{BinTree}(A) & \xrightarrow[\mathsf{iotrv} = \mathrm{int}_{[\langle\rangle, g]}]{} & A^\star \end{array}$$

where the function $g \colon A^\star \times A \times A^\star \to A^\star$ is defined by $g(\sigma, a, \tau) = \sigma \cdot a \cdot \tau$. Similarly, the function $h(\sigma, a, \tau) = a \cdot \sigma \cdot \tau$ is used in the definition of preorder traversal:

$$\begin{array}{ccc} 1+(\mathsf{BinTree}(A) \times A \times \mathsf{BinTree}(A)) & \xrightarrow{\mathrm{id}_1 + (\mathsf{potrv} \times \mathrm{id}_A \times \mathsf{potrv})} & 1+(A^\star \times A \times A^\star) \\ \downarrow{\scriptstyle [\mathsf{nil}, \mathsf{node}]} & & \downarrow{\scriptstyle [\langle\rangle, h]} \\ \mathsf{BinTree}(A) & \xrightarrow[\mathsf{potrv} = \mathrm{int}_{[\langle\rangle, h]}]{} & A^\star \end{array}$$

What we see is that the familiar pattern matching in inductive definitions fits perfectly in the initiality scheme, where the way the various patterns are handled corresponds to the algebra structure on the codomain of the function that is being defined.

It turns out that many such functional programs can be defined elegantly via initiality (and also finality); this style that is called 'origami programming' in [153]. Moreover, via uniqueness one can establish various properties about these programs. This has developed into an area of its own, which is usually referred to as the Bird–Meertens formalism – with its own peculiar notation and terminology; see [72] for an overview.

In Example 2.4.2 we have seen how functors on categories other than **Sets** can be useful, namely for describing multi-sorted signatures. The next example is another illustration of the usefulness of descriptions in terms of functors.

Example 2.4.6 (Refinement types) For a fixed set A we can form the functor $F(X) = 1 + (A \times X)$. The initial algebra of this functor is the set $A^\star$ of finite lists of elements of A, with algebra structure given by the empty list nil and the prefix operation cons in

$$1 + (A \times A^\star) \xrightarrow[\cong]{[\mathsf{nil}, \mathsf{cons}]} A^\star.$$

Via this initiality one can define for instance the length function $\mathsf{len} \colon A^\star \to \mathbb{Z}$. For some special reasons we take the integers $\mathbb{Z}$, instead of the natural numbers $\mathbb{N}$, as codomain type (as is common in computer science); the reasons will become clear later on. A definition of length by initiality requires that the set $\mathbb{Z}$ be equipped with an appropriate F-coalgebra structure:

$$\begin{array}{ccc} 1 + A \times A^\star & \xrightarrow{\mathrm{id}_1 + \mathrm{id}_A \times \mathsf{len}} & 1 + A \times \mathbb{Z} \\ {\scriptstyle [\mathsf{nil}, \mathsf{cons}]}\downarrow \cong & & \downarrow{\scriptstyle [0, \mathsf{S} \circ \pi_2]} \\ A^\star & \xrightarrow[\mathsf{len}]{} & \mathbb{Z} \end{array}$$

Commutation of this diagram corresponds to the expected defining equations for list-length:

$$\mathsf{len}(\mathsf{nil}) = 0 \quad \text{and} \quad \mathsf{len}(\mathsf{cons}(a, \sigma)) = \mathsf{S}(\mathsf{len}(\sigma)) = 1 + \mathsf{len}(\sigma).$$

One can also define an append map $\mathsf{app} \colon A^\star \times A^\star \to A^\star$ that concatenates two lists producing a new one, but this requires some care: by initiality one can define only maps of the form $A^\star \to X$, when X carries an F-algebra structure. The easiest way out is to fix a list $\tau \in A^\star$ and to define $\mathsf{app}(-, \tau) \colon A^\star \to A^\star$ by initiality in

$$\begin{array}{ccc} 1 + A \times A^\star & \xrightarrow{\mathrm{id}_1 + \mathrm{id}_A \times \mathsf{app}(-,\tau)} & 1 + A \times A^\star \\ {\scriptstyle [\mathsf{nil}, \mathsf{cons}]}\downarrow \cong & & \downarrow{\scriptstyle [\tau, \mathsf{cons}]} \\ A^\star & \xrightarrow[\mathsf{app}(-,\tau)]{} & A^\star \end{array}$$

Alternatively, one can define the append map app via currying, namely as map $A^\star \to (A^\star)^{A^\star}$. This will be left to the interested reader. But one can also use the stronger principle of 'induction with parameters'; see Exercise 2.5.5.

Now assume the set A carries a binary operation $+\colon A \times A \to A$. We would like to use it to define an operation sum on lists, namely as

$$\mathsf{sum}(\langle a_1, \ldots, a_n\rangle, \langle b_1, \ldots, b_n\rangle) = \langle a_1 + b_1, \ldots, a_n + b_n\rangle.$$

But this works only for lists of the same length!

Hence what we would like to have is a special type of lists of a certain length n. It can be formed as an inverse image:

$$\mathsf{len}^{-1}(\{n\}) = \{\sigma \in A^\star \mid \mathsf{len}(\sigma) = n\} = \begin{cases} A^n & \text{if } n \geq 0 \\ \emptyset & \text{otherwise.} \end{cases}$$

The type of the function sum can now be described accurately as $A^n \times A^n \to A^n$. We now consider the map $\mathsf{len}\colon A^\star \to \mathbb{Z}$ as a map in a slice category $\mathbf{Sets}/\mathbb{Z}$. Recall from Exercise 1.4.3 that this map $\mathsf{len}\colon A^\star \to \mathbb{Z}$ can be identified with the indexed collection $(\mathsf{len}^{-1}(\{n\}))_{n\in\mathbb{Z}}$ given by inverse images. An interesting question is: can we also define such maps like sum on dependent types like $\mathsf{len}^{-1}(\{n\})$ by initiality? This is relevant for dependently typed programming languages like *Agda*.[1]

In [45] a solution is given. We sketch the essentials, in quite general terms. Let $F\colon \mathbb{C} \to \mathbb{C}$ be an arbitrary functor on a category $\mathbb{C}$. Assume an F-algebra $b\colon F(B) \to B$; it plays the role of $\mathbb{Z}$ above, and the elements $x \in B$ are used to form a refinement of F's initial algebra. But first we transform F from an endofunctor on $\mathbb{C}$ into an endofunctor on the slice category $\mathbb{C}/B$. Intuitively this slice category contains objects of $\mathbb{C}$ indexed by B; see Exercise 1.4.3. Recall that objects of $\mathbb{C}/B$ are arrows with codomain B, and morphisms are commuting triangles.

We now define a functor $F^b\colon \mathbb{C}/B \to \mathbb{C}/B$ by

$$\begin{array}{ccc} (X \xrightarrow{f} B) & \longmapsto & (F(X) \xrightarrow{F(f)} F(B) \xrightarrow{b} B) \\ \left(\begin{array}{c} X \xrightarrow{h} Y \\ {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle g} \\ B \end{array}\right) & \longmapsto & \left(\begin{array}{c} F(X) \xrightarrow{F(h)} F(Y) \\ {\scriptstyle F(f)}\searrow \qquad \swarrow{\scriptstyle F(g)} \\ F(B) = F(B) \\ {\scriptstyle b}\searrow \quad \swarrow{\scriptstyle b} \\ B \end{array}\right) \end{array}$$

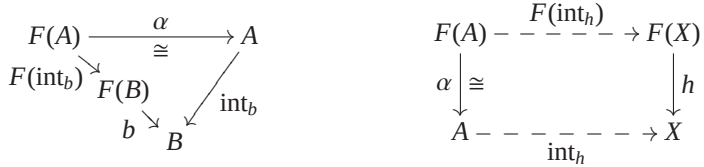
Next we assume an initial F-algebra $\alpha\colon F(A) \xrightarrow{\cong} A$. The algebra b gives rise to a homomorphism $\mathrm{int}_b\colon A \to B$ by initiality. This map is an object in the slice category $\mathbb{C}/B$. It can be understood as what is called a **refinement type** since it corresponds to an indexed family $(A_x)_{x\in B}$, where

[1] See http://wiki.portal.chalmers.se/agda.

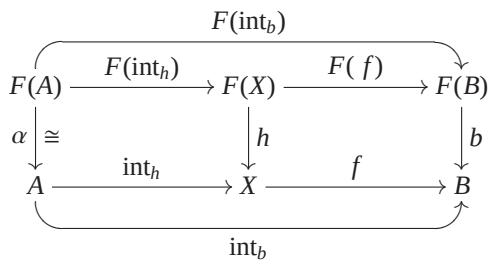
$A_x = \mathrm{int}_b^{-1}(\{x\}) = \{a \in A \mid \mathrm{int}_b(a) = x\}$. In fact, this type is also an initial algebra, but in a slice category, as observed in [45].

Claim. The refinement type $\mathrm{int}_b : A \to B$ in $\mathbb{C}/B$ carries the initial algebra of the functor $F^b : \mathbb{C}/B \to \mathbb{C}/B$ defined above.

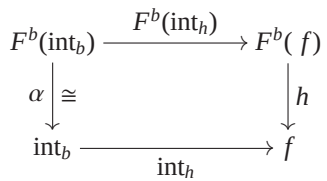
The truth of this claim is not hard to see but requires that we carefully keep track of the category that we work in (namely $\mathbb{C}$ or $\mathbb{C}/B$). We first have to produce an algebra structure $F^b(\mathrm{int}_b) \to \mathrm{int}_b$ in the slice category $\mathbb{C}/B$. When we work out what it can be we see that the map $\alpha : F(A) \xrightarrow{\cong} A$ is the obvious candidate, on the left in



Next, if we have an arbitrary F^b-coalgebra, say given by a map $h : F^b(X \xrightarrow{f} B) \longrightarrow (X \xrightarrow{f} B)$ in the slice category $\mathbb{C}/B$, then this h is a map $h : F(X) \to X$ in $\mathbb{C}$ satisfying $f \circ h = b \circ F(h)$. By initiality we get an algebra homomorphism $\mathrm{int}_h : A \to X$ as on the right above. We claim that the resulting interpretation map $\mathrm{int}_h : A \to X$ is a morphism $\mathrm{int}_b \to f$ in $\mathbb{C}/B$. This means that $f \circ \mathrm{int}_h = \mathrm{int}_b$, which follows by a uniqueness argument in



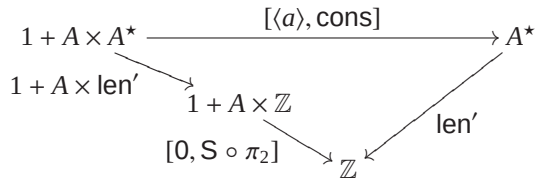
What we finally need to show is that we have a commuting diagram in $\mathbb{C}/B$ of the form



Commutation of this square in $\mathbb{C}/B$ amounts to commutation of the corresponding diagram in $\mathbb{C}$, which is the case by definition of int_h. The proof of uniqueness of this homomorphism of F^b-algebras $\mathrm{int}_b \to f$ is left to the reader.

At this stage we return to the length map $\mathsf{len}\colon A^\star \to \mathbb{Z}$ defined in the beginning of this example. We now use initiality in the slice category $\mathbf{Sets}/\mathbb{Z}$ to define a 'reverse cons' operation $\mathsf{snoc}\colon A^n \times A \to A^{n+1}$ that adds an element at the end of a list. We do so, as before, by fixing a parameter $a \in A$ and defining $\mathsf{snoc}(-, a)\colon A^n \to A^{n+1}$. We need to come up with an algebra of the endofunctor F^{len} on $\mathbf{Sets}/\mathbb{Z}$, where $F = 1 + (A \times -)$. The claim above says that $\mathsf{len}\colon A^\star \to \mathbb{Z}$ is the initial F^{len}-algebra.

As carrier of the algebra we take the object $\mathsf{len}'\colon A^\star \to \mathbb{Z}$ in $\mathbf{Sets}/\mathbb{Z}$ given by $\mathsf{len}'(\sigma) = \mathsf{len} - 1$. It is at this point that we benefit from using the integers instead of the naturals. The algebra map $F^{\mathsf{len}}(\mathsf{len}') \to \mathsf{len}'$ we use is



Here we use $\langle a \rangle = \mathsf{cons}(a, \mathsf{nil})$ for the singleton list. The triangle commutes because

$$\begin{aligned} \mathsf{len}'(\langle a \rangle) &= \mathsf{len}(\langle a \rangle) - 1 = 1 - 1 = 0 \\ \mathsf{len}'(\mathsf{cons}(a, \sigma)) &= \mathsf{len}(\mathsf{cons}(a, \sigma)) - 1 = \mathsf{len}(\sigma) + 1 - 1 = \mathsf{len}(\sigma) - 1 + 1 \\ &= \mathsf{S}(\mathsf{len}'(\sigma)) = (\mathsf{S} \circ \pi_2 \circ (A \times \mathsf{len}'))(a, \sigma). \end{aligned}$$

Hence by initiality of the map $\mathsf{len}\colon A^\star \to \mathbb{Z}$ there is unique algebra homomorphism $\mathsf{snoc}(-, a)\colon \mathsf{len} \to \mathsf{len}'$ in $\mathbf{Sets}/\mathbb{Z}$. This means: $\mathsf{len}' \circ \mathsf{snoc}(-, a) = \mathsf{len}$. Thus, for each $\sigma \in A^\star$,

$$\mathsf{len}(\sigma) = \mathsf{len}'(\mathsf{snoc}(a, \sigma)) = \mathsf{len}(\mathsf{snoc}(a, \sigma)) - 1,$$

i.e.

$$\mathsf{len}(\mathsf{snoc}(\sigma, a)) = \mathsf{len}(\sigma) + 1.$$

Alternatively, for each $n \in \mathbb{N}$ one has a typing:

$$A^n = \mathsf{len}^{-1}(\{n\}) \xrightarrow{\mathsf{snoc}} \mathsf{len}^{-1}(\{n+1\}) = A^{n+1}\ .$$

This concludes our refinement types example.

As we have seen in Example 2.4.4, an inductive definition of a function $f\colon \mathbb{N} \to X$ on the natural numbers requires two functions, $a\colon 1 \to X$ and $g\colon X \to X$. A **recursive** definition allows for an additional parameter, via a

function $h\colon X \times \mathbb{N} \to X$, determining $f(n+1) = h(f(n), n)$. The next result shows that recursion can be obtained via induction. A more general approach via comonads may be found in [455]. An alternative way to add parameters to induction occurs in Exercise 2.5.5.

Proposition 2.4.7 (Recursion) *An arbitrary initial algebra* $\alpha\colon F(A) \xrightarrow{\cong} A$ *satisfies the following recursion property: for each map* $h\colon F(X \times A) \to X$ *there is a unique map* $f\colon A \to X$ *making the following diagram commute*:

$$\begin{array}{ccc} F(A) & \xrightarrow{F\langle f, \mathrm{id}\rangle} & F(X \times A) \\ {\scriptstyle\alpha}\downarrow\cong & & \downarrow{\scriptstyle h} \\ A & \xrightarrow[f]{} & X \end{array}$$

Proof We first turn $h\colon F(X \times A) \to X$ into an algebra on $X \times A$, namely by taking $h' = \langle h, \alpha \circ F(\pi_2)\rangle\colon F(X \times A) \to X \times A$. It gives by initiality rise to a unique map $k\colon A \to X \times A$ with $k \circ \alpha = h' \circ F(k)$. Then $\pi_2 \circ k = \mathrm{id}$ by uniqueness of algebra maps $\alpha \to \alpha$:

$$\pi_2 \circ k \circ \alpha = \pi_2 \circ h' \circ F(k) = \alpha \circ F(\pi_2) \circ F(k) = \alpha \circ F(\pi_2 \circ k).$$

Hence we take $f = \pi_1 \circ k$. ❑

Adding parameters to the formulation of induction gives another strengthening; see Exercise 2.5.5.

So far we have seen only algebras of functors describing fairly elementary datatypes. We briefly mention some other less trivial applications.

- Joyal and Moerdijk [286] introduce the notion of a *Zermelo–Fraenkel algebra*, or *ZF-algebra*, with two operations for union and singleton. The usual system of Zermelo–Fraenkel set theory can then be characterised as the free ZF-algebra. Such a characterisation can be extended to ordinals.
- Fiore, Plotkin and Turi [136] describe how variable binding – such as $\lambda x.\, M$ in the lambda calculus – can also be captured via initial algebras, namely of suitable functors on categories of presheaves. An alternative approach, also via initiality, is described by Gabbay and Pitts [148], using set theory with atoms (or *urelements*). See also [270].
- So-called name-passing systems, like the π-calculus, are also modelled as coalgebras, namely for endofunctors on categories of presheaves, see [137, 97, 349, 435].

In the remainder of this section we shall briefly survey several ways to combine algebras and coalgebras. Such combinations are needed in descriptions

of more complicated systems involving data as well as behaviour. Later on, in Section 5.5, we shall study bialgebras more systematically, for operational semantics.

Additionally, so-called binary methods are problematic in a coalgebraic setting. These are (algebraic) operations like $X \times X \to X$ in which the state space X occurs multiple times (positively) in the domain. The name 'binary method' comes from object-oriented programming, where the status of such methods is controversial due to the typing problems they can cause; see [88]. They are also problematic in coalgebra, because they cannot be described as a coalgebra $X \to F(X)$. However, one may ask whether it makes sense in system theory to have an operation acting on two states, so that the problematic character of their representation need not worry us very much. But see Exercise 2.4.10 for a possible way to handle binary methods in a coalgebraic manner.

2.4.1 Bialgebras

A **bialgebra** consists of an algebra-coalgebra pair $F(X) \to X \to G(X)$ on a common state space X, where $F, G \colon \mathbb{C} \to \mathbb{C}$ are two endofunctors on the same category $\mathbb{C}$. We shall investigate them more closely in Section 5.5 using so-called distributive laws, following the approach of [452, 451, 58, 298].

Such structures are used in [69] to distinguish certain observer operations as coalgebraic operations within algebraic specification and to use these in a duality between reachability and observability (following the result of Kalman; see Exercise 2.5.15).

2.4.2 Dialgebras

A **dialgebra** consists of a mapping $F(X) \to G(X)$ where F and G are two functors $\mathbb{C} \to \mathbb{C}$. This notion generalises both algebras (for $G = \mathrm{id}$) and coalgebras (for $F = \mathrm{id}$). It was introduced in [191] and further studied for example in [138] in combination with 'laws' as a general concept of equation. In [392] a collection of such dialgebras $F_i(X) \to G_i(X)$ is studied, in order to investigate the commonalities between algebra and coalgebra, especially related to invariants and bisimulations.

2.4.3 Hidden Algebras

Hidden algebra is introduced in [164] as the 'theory of everything' in software engineering, combining the paradigms of object-oriented, logic, constraint and functional programming. A hidden algebra does not formally combine

algebras and coalgebras, as in bi-/di-algebras, but it uses an algebraic syntax to handle essentially coalgebraic concepts, such as behaviour and observational equivalence. A key feature of the syntax is the partition of sorts (or types) into visible and hidden ones. Typically, data structures are visible, and states are hidden. Elements of the visible sorts are directly observable and comparable, but observations about elements of the hidden sorts can be made only via the visible ones. This yields a notion of behavioural equivalence – first introduced by Reichel – expressed in terms of equality of contexts of visible sort. This is in fact bisimilarity, from a coalgebraic perspective; see [339, 98].

Because of the algebraic syntax of hidden algebras one cannot represent typically coalgebraic operations. But one can mimic them via subsorts. For instance, a coalgebraic operation $X \longrightarrow 1 + X$ can be represented as a 'partial' function $S \longrightarrow X$ for a subsort $S \subseteq X$. Similarly, an operation $X \longrightarrow X + X$ can be represented via two operations, $S_1 \longrightarrow X$ and $S_2 \longrightarrow X$, for subsorts $S_1, S_2 \subseteq X$ with $S_1 \cup S_2 = X$ and $S_1 \cap S_2 = \emptyset$. This quickly gets out of hand for more complicated operations, like the methods meth_i from (1.10), so that the naturality of coalgebraic representations is entirely lost. On the positive side, hidden algebras can handle binary methods without problems – see also Exercise 2.4.10.

2.4.4 Coalgebras as Algebras

In general, there is reasonable familiarity with algebra, but not (yet) with coalgebra. Therefore, people like to understand coalgebras as if they were algebras. Indeed, there are many connections. For example, in [55] it is shown that quite often the final coalgebra $Z \xrightarrow{\cong} F(Z)$ of a functor F can be understood as a suitable form of completion of the initial algebra $F(A) \xrightarrow{\cong} A$ of the same functor. Examples are the extended natural numbers $\mathbb{N} \cup \{\infty\}$ from Exercise 2.4.1 below, as completion of the initial algebra $\mathbb{N}$ of the functor $X \mapsto 1+X$, and the set A^∞ of finite and infinite sequences as completion of the initial algebra $A^\star$ with finite sequences only, of the functor $X \mapsto 1 + (A \times X)$.

Also, since a final coalgebra $\zeta\colon Z \xrightarrow{\cong} F(Z)$ is an isomorphism, one can consider its inverse $\zeta^{-1}\colon F(Z) \xrightarrow{\cong} Z$ as an algebra. This happens for example in the formalisation of 'coinductive types' (final coalgebras) in the theorem prover Coq[2] (and used for instance in [112, 225]). However, this may lead to rather complicated formulations of coinduction, distracting from the coalgebraic essentials. Therefore we like to treat coalgebra as a field of its own, and not in an artificial way as part of algebra.

[2] See https://coq.inria.fr/.

By now coalgebraic constructs are appearing in many languages for programming and proving; see e.g. [189, 415, 448, 1, 277, 278].

Exercises

2.4.1 Check that the set $\mathbb{N} \cup \{\infty\}$ of extended natural numbers is the final coalgebra of the functor $X \mapsto 1 + X$, as a special case of finality of A^∞ for $X \mapsto 1 + (A \times X)$. Use this fact to define appropriate addition and multiplication operations $(\mathbb{N} \cup \{\infty\}) \times (\mathbb{N} \cup \{\infty\}) \to \mathbb{N} \cup \{\infty\}$; see [411].

2.4.2 Define an appropriate size function $\mathsf{BinTree}(A) \to \mathbb{N}$ by initiality.

2.4.3 Consider the obvious functions $\mathsf{even}, \mathsf{odd} \colon \mathbb{N} \to 2 = \{0, 1\}$.

1. Describe the two algebra structures $1 + 2 \to 2$ for the functor $F(X) = 1 + X$ that define even and odd by initiality.
2. Define a single algebra $1 + (2 \times 2) \to 2 \times 2$ that defines the pair $\langle \mathsf{even}, \mathsf{odd} \rangle$ by mutual recursion, that is via $\mathsf{even}(n + 1) = \mathsf{odd}(n)$.

2.4.4 Show, dually to Proposition 2.1.5, that finite products $(1, \times)$ in a category $\mathbb{C}$ give rise to finite products in a category $\mathbf{Alg}(F)$ of algebras of a functor $F \colon \mathbb{C} \to \mathbb{C}$.

2.4.5 Define addition $+ \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, multiplication $\cdot \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and exponentiation $(-)^{(-)} \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ by initiality. *Hint*: Use the correspondence (2.11) and define these operations as function $\mathbb{N} \to \mathbb{N}^{\mathbb{N}}$. Alternatively, one can use Exercise 2.5.5 from the next section.

2.4.6 Complete the proof of Proposition 2.4.7.

2.4.7 ('Rolling lemma') Assume two endofunctors $F, G \colon \mathbb{C} \to \mathbb{C}$ on the same category. Let the composite $FG \colon \mathbb{C} \to \mathbb{C}$ have an initial algebra $\alpha \colon FG(A) \xrightarrow{\cong} A$.

1. Prove that also the functor $GF \colon \mathbb{C} \to \mathbb{C}$ has an initial algebra, with $G(A)$ as carrier.
2. Formulate and prove a dual result, for final coalgebras.

2.4.8 This exercise illustrates the analogy between the set $A^\star$ of finite sequences in **Sets** and the vector space $A^{\S}$ in **Vect**, following Exercise 2.2.12. Recall from Example 2.4.6 that the set $A^\star$ of finite lists of elements of A is the initial algebra of the functor $X \mapsto 1 + (A \times X)$.

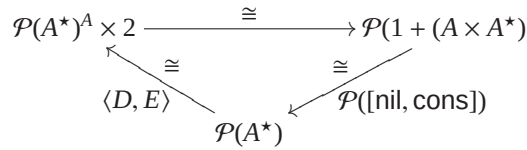
1. For an arbitrary vector space A, this same functor $1 + (A \times \mathrm{id})$, but considered as an endofunctor on **Vect**, can be rewritten as

$$\begin{aligned} 1 + (A \times X) &\cong A \times X && \text{because } 1 \in \mathbf{Vect} \text{ is initial object} \\ &\cong A + X && \text{because } \times \text{ and } + \text{ are the same in } \mathbf{Vect}. \end{aligned}$$

Prove that the initial algebra of this functor $A + \mathrm{id} : \mathbf{Vect} \to \mathbf{Vect}$ is the vector space $A^{\S}$ with insertion and shift maps $\mathsf{in} : A \to A^{\S}$ and $\mathsf{sh} : A^{\S} \to A^{\S}$ defined in Exercise 2.2.12.3.

2. Check that the behaviour formula from Exercise 2.2.12.4 for a system $A \xrightarrow{F} X \xrightarrow{G} X \xrightarrow{H} B$ is obtained as $H \circ \mathrm{int}_{[F,G]} : A^{\S} \to B$ using initiality.
3. Show that the assignment $A \mapsto A^{\S}$ yields a functor $\mathbf{Vect} \to \mathbf{Vect}$. *Hint*: Actually, this is a special case of the dual of Exercise 2.3.8.

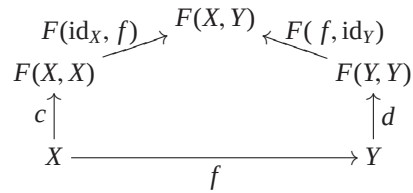
2.4.9 Use Exercise 2.1.10 to show that there is a commuting diagram of isomorphisms



where the coalgebra $\langle D, E \rangle$ is the final Brzozowski deterministic automaton structure from Corollary 2.3.6.2, and $[\mathsf{nil}, \mathsf{cons}]$ is the initial list algebra (from Example 2.4.6).

2.4.10 Suppose we have a 'binary method', say of the form $m : X \times X \times A \longrightarrow 1 + (X \times B)$. There is a well-known trick from [143] to use a functorial description also in such cases, namely by separating positive and negative occurrences. This leads to a functor of the form $F : \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbb{C}$, which in this case would be $(Y, X) \mapsto (1 + (X \times B))^{Y \times A}$.

In general, for a functor $F : \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbb{C}$ define an F-coalgebra to be a map of the form $c : X \to F(X, X)$. A homomorphism from $c : X \to F(X, X)$ to $d : Y \to F(Y, Y)$ is then a map $f : X \to Y$ in $\mathbb{C}$ making the following pentagon commute:



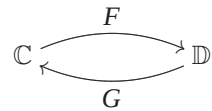
1. Elaborate what this means for the above example $(Y, X) \mapsto (1 + (X \times B))^{Y \times A}$.

2. Prove that these coalgebras and their homomorphisms form a category.

(Such generalised coalgebras are studied systematically in [444].)

2.5 Adjunctions, Cofree Coalgebras, Behaviour-Realisation

The concept of an adjunction may be considered as one of the basic contributions of the theory of categories. It consists of a pair of functors going in opposite direction,



satisfying certain properties. An adjunction is a fundamental notion, occurring in many, many situations. Identifying an adjunction is useful, because it captures much information and yields additional insights such as the 'left' adjoint functor preserves coproducts, and the 'right' adjoint preserves products. This is the good news. The bad news is that the notion of adjunction is considered to be a difficult one. It certainly requires some work and experience to fully appreciate its usefulness. Much of this text can be read without knowledge of adjunctions. However, there are certain results which can best be organised in terms of adjunctions. Therefore we include an introduction to adjunctions and apply it to three standard examples in the theory of coalgebras and algebras, namely behaviour-realisation, cofree coalgebras and liftings of adjunctions to categories of (co)algebras.

We start with 'baby adjunctions', namely Galois connections. Consider two posets C and D as categories, with monotone functions $f\colon C \to D$ and $g\colon D \to C$ between them, in opposite direction. They can be regarded as functors by Example 1.4.4.2. These functions form a **Galois connection** if for each $x \in C$ and $y \in D$ one has $f(x) \leq y$ in D if and only if $x \leq g(y)$ in C. We like to write this as a bijective correspondence:

$$\frac{f(x) \;\leq\; y}{x \;\leq\; g(y)}\,.$$

In this situation one calls f the left or lower adjoint, and g the right or upper adjoint.

With these correspondences it is easy to show that the left adjoint f preserves joins $\vee$ that exist in C, i.e. that $f(x_1 \vee x_2) = f(x_1) \vee f(x_2)$. This is done by

showing that $f(x_1 \vee x_2)$ is the least upperbound in D of $f(x_1)$ and $f(x_2)$: for any $y \in D$,

$$\dfrac{\dfrac{\dfrac{f(x_1 \vee x_2) \;\leq\; y}{x_1 \vee x_2 \;\leq\; g(y)}}{x_1 \;\leq\; g(y)} \qquad \dfrac{x_2 \;\leq\; g(y)}{f(x_2) \;\leq\; y}}{f(x_1) \;\leq\; y} \;.$$

This says that $f(x_1 \vee x_2) \leq y$ if and only if both $f(x_1) \leq y$ and $f(x_2) \leq y$, and thus that $f(x_1 \vee x_2)$ is the join of $f(x_1)$ and $f(x_2)$.

The notion of adjunction is defined more generally for functors between categories. It involves a bijective correspondence like above, together with certain technical naturality conditions. These side conditions make the definition a bit complicated but are usually trivial in concrete examples. Therefore, the bijective correspondence is what should be kept in mind.

Definition 2.5.1 Consider two categories $\mathbb{C}$ and $\mathbb{D}$ with functors $F\colon \mathbb{C} \to \mathbb{D}$ and $G\colon \mathbb{D} \to \mathbb{C}$ between them. These functors form an **adjunction**, written as $F \dashv G$, if for all objects $X \in \mathbb{C}$ and $Y \in \mathbb{D}$ there are bijective correspondences between morphisms

$$\dfrac{F(X) \longrightarrow Y}{X \longrightarrow G(Y)} \qquad \begin{array}{l} \text{in } \mathbb{D} \\ \text{in } \mathbb{C} \end{array}$$

which are 'natural' in X and Y. In this case one says that F is the left adjoint, and G the right adjoint. The map $X \to G(Y)$ corresponding to $F(X) \to Y$ (and vice-versa) is sometimes called the transpose.

We write this correspondences as functions $\psi\colon \mathbb{D}(F(X), Y) \xrightarrow{\cong} \mathbb{C}(X, G(Y))$. The naturality requirement then says that for morphisms $f\colon X' \to X$ in $\mathbb{C}$ and $g\colon Y \to Y'$ in $\mathbb{D}$ one has, for $h\colon F(X) \to Y$,

$$G(g) \circ \psi(h) \circ f \;=\; \psi(g \circ h \circ F(f)).$$

Since morphisms in poset categories are so trivial, the naturality requirement disappears in the definition of a Galois connection.

There are several equivalent ways to formulate the concept of an adjunction, using for example unit and counit natural transformations, or freeness (for left adjoints) or cofreeness (for right adjoints). We shall describe one alternative, namely the unit-and-counit formulation, in Exercise 2.5.7, and refer the interested reader to [344, chapter IV] for more information. At this stage we are mainly interested in a workable formulation.

We continue with an elementary example. It illustrates that adjunctions involve canonical translations back and forth, which can be understood as

minimal (or, more technically, free) constructions for left adjoints, and as maximal (or cofree) for right adjoints. Earlier, Exercise 1.4.4 described the set of finite sequences $A^\star$ as a *free* monoid on a set A. Below in Exercise 2.5.1 this will be rephrased in terms of an adjunction; it gives a similar minimality phenomenon.

Example 2.5.2 (From sets to preorders) Recall from Example 1.4.2.2 the definition of the category **PreOrd** with preorders $(X, \leq)$ as objects – where $\leq$ is a reflexive and transitive order on X – and monotone functions between them as morphisms. There is then an obvious forgetful functor $U\colon$ **PreOrd** $\to$ **Sets**, given by $(X, \leq) \mapsto X$. It maps a preorder to its underlying set and forgets about the order. This example shows that the forgetful functor U has both a left and a right adjoint.

The left adjoint $D\colon$ **Sets** $\to$ **PreOrd** sends an arbitrary set A to the 'discrete' preorder $D(A) = (A, \mathrm{Eq}(A))$ obtained by equipping A with the equality relation $\mathrm{Eq}(A) = \{(a, a) \mid a \in A\}$. A key observation is that for a preorder $(X, \leq)$ any function $f\colon A \to X$ is automatically a monotone function $(A, \mathrm{Eq}(A)) \to (X, \leq)$. This means that there is a trivial (identity) bijective correspondence:

$$\begin{array}{lll} D(A) = (A, \mathrm{Eq}(A)) \xrightarrow{\quad f \quad} (X, \leq) & & \text{in } \mathbf{PreOrd} \\ \hline\hline A \xrightarrow[\quad f \quad]{} X = U(X, \leq) & & \text{in } \mathbf{Sets}. \end{array}$$

This yields an adjunction $D \dashv U$.

The forgetful functor $U\colon$ **PreOrd** $\to$ **Sets** has not only a left adjoint, via the discrete, minimal order $=$, but also a right adjoint, involving the indiscrete, maximal order: there is a functor $I\colon$ **Sets** $\to$ **PreOrd** that equips a set A with the indiscrete preorder $I(A) = (A, \top_{A\times A})$ with 'truth' relation $\top_{A\times A} = \{(a, a') \mid a, a' \in A\} = (A \times A \subseteq A \times A)$, in which all elements of A are related. Then, for a preorder $(X, \leq)$, any function $X \to A$ is automatically a monotone function $(X, \leq) \to (A, \top_{A\times A})$. Hence we again have trivial correspondences:

$$\begin{array}{lll} (X, \leq) \xrightarrow{\quad f \quad} (A, \top_{A\times A}) = I(A) & & \text{in } \mathbf{Sets} \\ \hline\hline U(X, \leq) = X \xrightarrow[\quad f \quad]{} A & & \text{in } \mathbf{PreOrd} \end{array}$$

yielding an adjunction $U \dashv I$.

This situation 'discrete $\dashv$ forgetful $\dashv$ indiscrete' is typical; see Exercise 2.5.6 below.

Next we shall consider examples of adjunctions in the context of coalgebras. The first result describes 'cofree' coalgebras: it gives a canonical construction of a coalgebra from an arbitrary set. Its proof relies on Theorem 2.3.9, which

is as yet unproven. The proof shows that actually checking that one has an adjunction can be quite a bit of work. Indeed, the notion of adjunction combines much information.

Proposition 2.5.3 *For a finite Kripke polynomial functor* $F\colon \mathbf{Sets} \to \mathbf{Sets}$, *the forgetful functor* $U\colon \mathbf{CoAlg}(F) \to \mathbf{Sets}$ *has a right adjoint. It sends a set* A *to the carrier of the final coalgebra of the functor* $A \times F(-)$.

Proof Given F, consider the functor $F'\colon \mathbf{Sets} \to \mathbf{Sets}$ given by $A \mapsto A \times F(-)$. Then F' is also a finite Kripke polynomial functor. Hence, by Theorem 2.3.9 it has a final coalgebra $\zeta^A = \langle \zeta_1^A, \zeta_2^A \rangle\colon \widehat{A} \xrightarrow{\cong} A \times F(\widehat{A})$, which we use to define an F-coalgebra:

$$G(A) \stackrel{\text{def}}{=} \Big(\widehat{A} \xrightarrow{\zeta_2^A} F(\widehat{A})\Big).$$

We first show that G extends to a functor $\mathbf{Sets} \to \mathbf{CoAlg}(F)$. Let $f\colon A \to B$ therefore be an arbitrary function. We define G on f as a function $G(f)\colon \widehat{A} \to \widehat{B}$ by finality, in

$$\begin{array}{ccc}
B \times F(\widehat{A}) & \xrightarrow{\mathrm{id}_B \times F(G(f))} & B \times F(\widehat{B}) \\
{\scriptstyle f \times \mathrm{id}_{F(\widehat{A})}}\uparrow & & \uparrow \\
A \times F(\widehat{A}) & & \cong \uparrow {\scriptstyle \zeta^B} \\
{\scriptstyle \zeta^A}\uparrow \cong & & \\
\widehat{A} & \xrightarrow[G(f)]{} & \widehat{B}
\end{array}$$

Clearly,

$$\begin{aligned}
\zeta_2^B \circ G(f) &= \pi_2 \circ \zeta^B \circ G(f) \\
&= \pi_2 \circ (\mathrm{id}_B \times F(G(f))) \circ (f \times \mathrm{id}_{F(\widehat{A})}) \circ \zeta^A \\
&= F(G(f)) \circ \pi_2 \circ \zeta^A \\
&= F(G(f)) \circ \zeta_2^A.
\end{aligned}$$

Hence $G(f)$ is a homomorphism of coalgebras $G(A) \to G(B)$, as required. By uniqueness one shows that G preserves identities and compositions. This requires a bit of work but is not really difficult.

The adjunction $U \dashv G$ that we want requires a (natural) bijective correspondence:

$$U\begin{pmatrix} F(X) \\ {\scriptstyle c}\uparrow \\ X \end{pmatrix} = X \xrightarrow{\quad g \quad} A \qquad \text{in } \mathbf{Sets}$$

$$\Downarrow$$

$$\begin{pmatrix} F(X) \\ {\scriptstyle c}\uparrow \\ X \end{pmatrix} \xrightarrow[h]{} \begin{pmatrix} F(\widehat{A}) \\ \uparrow{\scriptstyle \zeta_2^A} \\ \widehat{A} \end{pmatrix} = G(A) \qquad \text{in } \mathbf{CoAlg}(F).$$

That is,

$$\begin{array}{ccc} X & \xrightarrow{\quad g \quad} & A \\ \Downarrow & & \\ F(X) & \xrightarrow[F(h)]{} & F(\widehat{A}) \\ {\scriptstyle c}\uparrow & & \uparrow{\scriptstyle \zeta_2^A} \\ X & \xrightarrow[h]{} & \widehat{A} \end{array}.$$

It is obtained as follows.

- Given a function $g\colon X \to A$, we can define $\overline{g}\colon X \to \widehat{A}$ by finality:

$$\begin{array}{ccc} A \times F(X) & \overset{\mathrm{id}_A \times F(\overline{g})}{- - - - \to} & A \times F(\widehat{A}) \\ {\scriptstyle \langle g, c\rangle}\uparrow & & \cong \uparrow{\scriptstyle \zeta^A = \langle \zeta_1^A, \zeta_2^A\rangle} \\ X & \underset{\overline{g}}{- - - - \to} & \widehat{A} \end{array}$$

  Then $\overline{g}$ is a homomorphism of coalgebras $c \to G(A)$ since

$$\zeta_2^A \circ \overline{g} = \pi_2 \circ (\mathrm{id}_A \times F(\overline{g})) \circ \langle g, c\rangle = F(\overline{g}) \circ \pi_2 \circ \langle g, c\rangle = F(\overline{g}) \circ c.$$

- Conversely, given a homomorphism $h\colon c \to G(A)$, take $\overline{h} = \zeta_1^A \circ h\colon X \to A$.

In order to complete the proof we still have to show bijectivity (i.e. $\overline{\overline{g}} = g$ and $\overline{\overline{h}} = h$) and naturality. The latter is left to the interested reader, but

$$\overline{\overline{g}} = \zeta_1^A \circ \overline{g} = \pi_1 \circ \zeta^A \circ \overline{g} = \pi_1 \circ (\mathrm{id}_A \times F(\overline{g})) \circ \langle g, c\rangle = \pi_1 \circ \langle g, c\rangle = g.$$

The second equation $\overline{\overline{h}} = h$ follows by uniqueness: $\overline{\overline{h}}$ is by construction the unique homomorphism k with $\zeta^A \circ k = (\mathrm{id}_A \times F(k)) \circ \langle \overline{h}, c\rangle$, i.e. with $\zeta^A \circ k = (\mathrm{id}_A \times F(k)) \circ \langle \zeta_1^A \circ h, c\rangle$. Since h is by assumption a homomorphism of coalgebras $c \to G(A)$, it also satisfies this condition. ❑

Dual to this result it makes sense to consider for algebras the *left* adjoint to a forgetful functor $\mathbf{Alg}(F) \to \mathbf{Sets}$. This gives so-called **free algebras**. They can be understood as term algebras built on top of a given collection of variables.

Since an adjunction $F \dashv G$ involves two translations $X \mapsto F(X)$ and $Y \mapsto G(Y)$ in opposite directions, one can translate objects 'back and forth', namely as $X \mapsto GF(X)$ and $Y \mapsto FG(Y)$. There are then canonical 'comparison' maps $\eta\colon X \to GF(X)$, called **unit**, and $\varepsilon\colon FG(Y) \to Y$, called **counit**.

In the context of the adjunction from the previous theorem, the unit η is a homomorphism from a coalgebra $X \to F(X)$ to the cofree coalgebra $\widehat{X} \to F(\widehat{X})$ on its carrier. It is obtained by finality. And the counit ε maps the carrier $\widehat{A}$ of a cofree coalgebra back to the original set A. It is ζ_1^A in the notation of the proof.

Using the notation $\psi\colon \mathbb{D}(F(X), Y) \xrightarrow{\cong} \mathbb{C}(X, G(Y))$ from Definition 2.5.1, the unit $\eta_X\colon X \to GF(X)$ and $\varepsilon_Y\colon FG(Y) \to Y$ maps can be defined as

$$\eta_X = \psi(\mathrm{id}_{F(X)}), \qquad \varepsilon_Y = \psi^{-1}(\mathrm{id}_{G(Y)}). \tag{2.30}$$

Using the naturality of ψ one easily checks that for arbitrary $f\colon X \to X'$ in $\mathbb{C}$ and $g\colon Y \to Y'$ in $\mathbb{D}$, the following diagrams commute:

$$\begin{array}{ccc} X & \xrightarrow{\eta_X} & GF(X) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle GF(f)} \\ X' & \xrightarrow[\eta_{X'}]{} & GF(X') \end{array} \qquad\qquad \begin{array}{ccc} FG(Y) & \xrightarrow{\varepsilon_Y} & Y \\ {\scriptstyle FG(g)}\downarrow & & \downarrow{\scriptstyle g} \\ FG(Y') & \xrightarrow[\varepsilon_{Y'}]{} & Y' \end{array}$$

This leads to the following fundamental notion in category theory: a natural transformation. It is a mapping between functors.

Definition 2.5.4 Let $\mathbb{C}$, $\mathbb{D}$ be two categories, with two (parallel) functors $H, K\colon \mathbb{C} \to \mathbb{D}$ between them. A **natural transformation** α from H to K consists of a collection of morphisms $\alpha_X\colon H(X) \to K(X)$ in $\mathbb{D}$, indexed by objects $X \in \mathbb{C}$ satisfying the naturality requirement: for each morphism $f\colon X \to Y$ in $\mathbb{C}$, the following rectangle commutes:

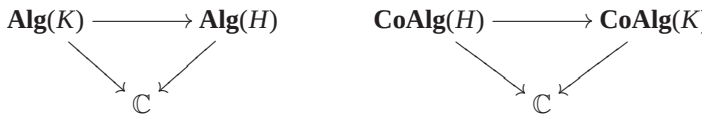
$$\begin{array}{ccc} H(X) & \xrightarrow{\alpha_X} & K(X) \\ {\scriptstyle H(f)}\downarrow & & \downarrow{\scriptstyle K(f)} \\ H(Y) & \xrightarrow[\alpha_Y]{} & K(Y) \end{array}$$

In this case one often writes $\alpha\colon H \Rightarrow K$, with double arrow.

The next basic result illustrates the relevance of natural transformations in the theory of algebras and coalgebras.

Proposition 2.5.5 *Consider a natural transformation $\alpha\colon H \Rightarrow K$ as defined above, for two endofunctors $H, K\colon \mathbb{C} \to \mathbb{C}$. This α induces translation*

functors between the corresponding categories of algebras and coalgebras, in commuting triangles:



These (horizontal) functors are given on objects by pre- and post-composition:

$$\begin{pmatrix} K(X) \\ \downarrow b \\ X \end{pmatrix} \longmapsto \begin{pmatrix} H(X) \\ \downarrow b \circ \alpha_X \\ X \end{pmatrix} \qquad \begin{pmatrix} H(X) \\ \uparrow c \\ X \end{pmatrix} \longmapsto \begin{pmatrix} H(X) \\ \uparrow \alpha_X \circ c \\ X \end{pmatrix}$$

On morphisms these functors are simply $f \mapsto f$.

Proof We shall write $F\colon \mathbf{Alg}(K) \to \mathbf{Alg}(H)$ for the mapping defined above. It is a well-defined functor, since if $f\colon X \to X'$ is a map of K-algebras $b \to b'$, then f is also a map of H-algebras $F(b) \to F(b')$, since

$$\begin{aligned} F(b') \circ H(f) &= b' \circ \alpha_X \circ H(f) \\ &= b' \circ K(f) \circ \alpha_X && \text{by naturality} \\ &= f \circ b \circ \alpha_X && \text{since } f \text{ is a map of } K\text{-algebras} \\ &= f \circ F(b). \end{aligned}$$

❑

For each set X there is an obvious map $\rho_X\colon X^\star \to \mathcal{P}_{\text{fin}}(X)$ mapping a sequence $\langle x_1, \ldots, x_n \rangle$ to the set $\{x_1, \ldots, x_n\}$ of elements involved – which may have size less than n in case duplicate elements occur. This operation is 'natural', also in a technical sense: for a function $f\colon X \to Y$ one has $\mathcal{P}_{\text{fin}}(f) \circ \rho_X = \rho_Y \circ f^\star$, since

$$\begin{aligned} (\rho_Y \circ f^\star)(\langle x_1, \ldots, x_n \rangle) &= \rho_Y(\langle f(x_1), \ldots, f(x_n) \rangle) \\ &= \{f(x_1), \ldots, f(x_n)\} \\ &= \mathcal{P}_{\text{fin}}(f)(\{x_1, \ldots, x_n\}) \\ &= (\mathcal{P}_{\text{fin}}(f) \circ \rho_X)(\langle x_1, \ldots, x_n \rangle). \end{aligned}$$

In the reverse direction $\mathcal{P}_{\text{fin}}(X) \to X^\star$ one can always choose a way to turn a finite set into a list, but there is no natural way to do this. More examples and non-examples are described at the end of Section 4.1. The idea is that natural transformations describe uniform mappings, such as given for instance by terms; see Exercise 2.5.17.

With this definition and notation we can write the unit and counit of an adjunction, from (2.30), as natural transformations $\eta\colon \mathrm{id}_{\mathbb{C}} \Rightarrow GF$ and $\varepsilon\colon FG \Rightarrow \mathrm{id}_{\mathbb{D}}$. The closely related notion of **equivalence** of categories is sometimes useful. It is an adjunction in which both the unit and counit are isomorphisms. This is a weaker notion than isomorphism of categories.

We shall consider another example of an adjunction which is typical in a coalgebraic setting, namely a so-called *behaviour-realisation* adjunction. Such adjunctions were first recognised in the categorical analysis of mathematical system theory; see [160–162]. Here we give a simple version using the deterministic automata introduced in Section 2.2. This requires two extensions of what we have already seen: (1) homomorphisms between these automata which allow variation in the input and output sets and (2) automata with initial states.

Definition 2.5.6 We write **DA** for a category of deterministic automata. It has the following:

objects deterministic automata $\langle \delta, \epsilon \rangle \colon X \to X^A \times B$ with an initial state $x_0 \in X$

morphisms from $\langle X \xrightarrow{\langle \delta, \epsilon \rangle} X^A \times B, x_0 \in X \rangle$ to $\langle Y \xrightarrow{\langle \delta', \epsilon' \rangle} Y^C \times D, y_0 \in Y \rangle$ consist of a triple of functions:

$$A \xleftarrow{f} C, \qquad B \xrightarrow{g} D, \qquad X \xrightarrow{h} Y,$$

with for all $x \in X$ and $c \in C$,

$$\begin{aligned} \delta'(h(x))(c) &= h(\delta(x)(f(c))) \\ \epsilon'(h(x)) &= g(\epsilon(x)) \\ h(x_0) &= y_0. \end{aligned}$$

The identity morphisms in **DA** are of the form $(\mathrm{id}, \mathrm{id}, \mathrm{id})$. The composition of (f_1, g_1, h_1) followed by (f_2, g_2, h_2) is $(f_1 \circ f_2, g_2 \circ g_1, h_2 \circ h_1)$. Note the reversed order in the first component.

The first two equations express that h is a coalgebra homomorphism from the automaton $\langle \mathrm{id}_X^f \circ \delta, g \circ \epsilon \rangle \colon X \to X^C \times D$, translated via (f, g), to the automaton $\langle \delta', \epsilon' \rangle \colon X \to X^C \times D$. Here we use the exponent notation id_X^f for functions from (2.12). The third equation simply says that the initial state is preserved.

We also introduce a category of deterministic behaviours. Its objects are elements of the final coalgebras for deterministic automata; see Proposition 2.3.5.

Definition 2.5.7 Form the category **DB** with

objects functions of the form $\varphi \colon A^\star \to B$

morphisms from $A^\star \xrightarrow{\varphi} B$ to $C^\star \xrightarrow{\psi} D$ are pairs of functions

$$A \xleftarrow{f} C, \qquad B \xrightarrow{g} D$$

with

$$g \circ \varphi \circ f^\star = \psi.$$

That is, for all $\langle c_1, \ldots, c_n \rangle \in C^\star$,

$$g(\varphi(\langle f(c_1), \ldots, f(c_n)\rangle)) = \psi(\langle c_1, \ldots, c_n \rangle).$$

The maps $(\mathrm{id}, \mathrm{id})$ are identities in **DB**. And composition in **DB** is given by $(f_2, g_2) \circ (f_1, g_1) = (f_1 \circ f_2, g_2 \circ g_1)$.

The behaviour-realisation adjunction for deterministic automata yields a canonical way to translate back-and-forth between deterministic automata and behaviours. It looks as follows.

Proposition 2.5.8 *There are a behaviour functor $\mathcal{B}$ and a realisation functor $\mathcal{R}$ in an adjunction:*

$$\begin{array}{ccc} & \mathbf{DA} & \\ \mathcal{B} & \dashv & \mathcal{R} \\ & \mathbf{DB} & \end{array}$$

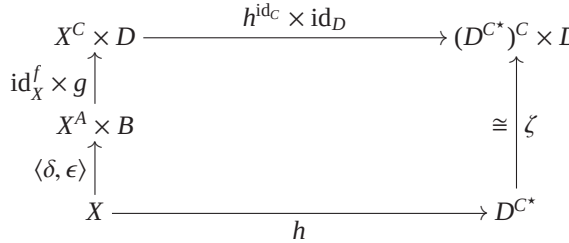
Proof We sketch the main lines and leave many details to the interested reader. The behaviour functor $\mathcal{B}\colon \mathbf{DA} \to \mathbf{DB}$ maps an automaton $\langle \delta, \epsilon \rangle \colon X \to X^A \times B$ with initial state $x_0 \in X$ to the behaviour $\mathrm{beh}_{\langle \delta, \epsilon \rangle}(x_0) \in B^{A^\star}$ of this initial state; see Proposition 2.3.5. On morphisms, it is $\mathcal{B}(f, g, h) = (f, g)$. This is well defined because $g \circ \mathrm{beh}(x) \circ f^\star = \mathrm{beh}'(h(x))$, as is checked easily by induction.

In the other direction, the realisation functor $\mathcal{R}\colon \mathbf{DB} \to \mathbf{DA}$ sends a behaviour $\psi\colon C^\star \to D$ to the final deterministic automaton $\zeta\colon D^{C^\star} \xrightarrow{\cong} (D^{C^\star})^C \times D$ described in Proposition 2.3.5, with ψ as initial state. The associated behaviour function beh_ζ is then given by $\mathrm{beh}_\zeta(\varphi) = \varphi$. On morphisms, $\mathcal{R}$ is defined as $\mathcal{R}(f, g) = (f, g, g^{f^\star})$.

The adjunction $\mathcal{B} \dashv \mathcal{R}$ is established by the bijective correspondence:

$$\frac{\mathcal{B}\Big(X \xrightarrow{\langle \delta, \epsilon \rangle} X^A \times B, x_0 \in X\Big) \xrightarrow{(f, g)} \Big(C^\star \xrightarrow{\psi} D\Big)}{\Big(X \xrightarrow{\langle \delta, \epsilon \rangle} X^A \times B, x_0 \in X\Big) \xrightarrow{(f, g, h)} \mathcal{R}\Big(C^\star \xrightarrow{\psi} D\Big)}.$$

This correspondence exists because the function h below the double line is uniquely determined by finality in the following diagram:



It satisfies $h(x_0) = \psi$ if and only if $g \circ \mathrm{beh}_{\langle \delta, \epsilon \rangle}(x_0) \circ f^\star = \psi$. ❑

Several alternative versions of this behaviour-realisation adjunction for deterministic automata are described in the exercises below. Such adjunctions have also been studied in more abstract settings; see for example [48] and [289].

One interesting aspect of the behaviour-realisation adjunction is that it provides a setting in which to (categorically) study various process operators. For instance, one can consider several ways to put deterministic automata in parallel. For automata $M_i = \langle X_i \xrightarrow{\langle \delta_i, \epsilon_i \rangle} X_i^{A_i} \times B_i, x_i \in X \rangle$, one can define new automata:

$$\begin{aligned} M_1 \mid M_2 &= \langle X_1 \times X_2 \xrightarrow{\langle \delta_\mid, \epsilon_\mid \rangle} (X_1 \times X_2)^{A_1 + A_2} \times (B_1 \times B_2), (x_0, x_1) \rangle \\ &\quad \text{where } \delta_\mid(y_1, y_2)(\kappa_1(a)) = (\delta_1(y_1)(a), y_2) \\ &\qquad\quad \delta_\mid(y_1, y_2)(\kappa_2(a)) = (y_1, \delta_2(y_2)(a)) \\ &\qquad\quad \epsilon_\mid(y_1, y_2) = (\epsilon_1(y_1), \epsilon_2(y_2)). \\ M_1 \otimes M_2 &= \langle X_1 \times X_2 \xrightarrow{\langle \delta_\otimes, \epsilon_\otimes \rangle} (X_1 \times X_2)^{A_1 \times A_2} \times (B_1 \times B_2), (x_0, x_1) \rangle \\ &\quad \text{where } \delta_\otimes(y_1, y_2)(a_1, a_2) = (\delta_1(y_1)(a), \delta_2(y_2)(a)) \\ &\qquad\quad \epsilon_\otimes(y_1, y_2) = (\epsilon_1(y_1), \epsilon_2(y_2)). \end{aligned}$$

The first composition of automata involves transitions in each component automaton separately, whereas the second composition combines transitions.

Both these definitions yield a so-called symmetric monoidal structure on the category **DA** of deterministic automata. Similar structure can be defined on the associated category **DB** of behaviours, in such a way that the behaviour functor $\mathcal{B}$: **DA** $\to$ **DB** preserves this structure. Thus, complex behaviour can be obtained from more elementary building blocks.

There is a line of research studying such 'process categories' with operations that are commonly used in process calculi as appropriate categorical structure; see for instance [216, 290, 213, 417, 314, 467, 53, 207, 202, 204].

This section concludes with a result about lifting adjunctions to categories of algebras; see [218, theorem 2.14]. It is purely categorical abstract nonsense

and does not talk about concrete categories or functors. In dual form it yields a lifting to categories of coalgebras.

Theorem 2.5.9 *Consider a natural transformation $\alpha\colon SF \Rightarrow FT$ in a situation:*

$$\begin{array}{ccc} \mathbb{A} & \xrightarrow{S} & \mathbb{A} \\ {\scriptstyle F}\uparrow & \swarrow\!\!\searrow\ {\scriptstyle \alpha} & \uparrow{\scriptstyle F} \\ \mathbb{B} & \xrightarrow[T]{} & \mathbb{B} \end{array} \tag{2.31}$$

It induces a lifting of F to a functor $\mathbf{Alg}(F)$ *in*

$$\begin{array}{ccc} \mathbf{Alg}(T) & \xrightarrow{\qquad\mathbf{Alg}(F)\qquad} & \mathbf{Alg}(S) \\ (T(X) \xrightarrow{a} X) & \longmapsto & (SF(X) \xrightarrow{\alpha_X} FT(X) \xrightarrow{F(a)} F(X)). \end{array}$$

If α is an isomorphism, then a right adjoint G to F induces a right adjoint $\mathbf{Alg}(G)$ *to* $\mathbf{Alg}(F)$ *in*

$$\begin{array}{ccc} \mathbb{A} & \xleftarrow{\qquad U\qquad} & \mathbf{Alg}(S) \\ {\scriptstyle F}\left(\dashv\right){\scriptstyle G} & & \mathbf{Alg}(F)\left(\dashv\right)\mathbf{Alg}(G) \\ \mathbb{B} & \xleftarrow[U]{} & \mathbf{Alg}(T) \end{array}$$

where the Us are forgetful functors. The functor $\mathbf{Alg}(G)$ *arises from $\beta\colon TG \Rightarrow GS$, the adjoint transpose of $FTG \cong SFG \overset{S\varepsilon}{\Longrightarrow} S$.*

Proof Of course the functor $\mathbf{Alg}(F)$ is defined on morphisms as $f \mapsto F(f)$. We check that this is well defined: for a homomorphism of T-algebras $(T(X) \xrightarrow{a} X) \xrightarrow{f} (T(Y) \xrightarrow{b} Y)$ we obtain that $F(f)$ is a homomorphism between the corresponding S-algebras in

$$\begin{array}{ccc} SF(X) & \xrightarrow{SF(f)} & SF(Y) \\ {\scriptstyle \alpha_X}\downarrow & & \downarrow{\scriptstyle \alpha_Y} \\ FT(X) & \xrightarrow{FT(f)} & FT(Y) \\ {\scriptstyle F(a)}\downarrow & & \downarrow{\scriptstyle F(b)} \\ F(X) & \xrightarrow[F(f)]{} & F(Y) \end{array}$$

The upper square commutes by naturality of α, and the lower square because f is a homomorphism of T-algebras.

Next we assume that α is an isomorphism, and write β_Y, where $Y \in \mathbb{A}$, for the following map in $\mathbb{B}$:

$$\beta_Y = \Big(TG(Y) \xrightarrow{\eta_{TG(Y)}} GFTG(Y) \xrightarrow{G(\alpha^{-1}_{G(Y)})} GSFG(Y) \xrightarrow{GS(\varepsilon_Y)} GS(Y)\Big).$$

Following the construction of the functor $\mathbf{Alg}(F)$, this natural transformation β induces a functor $\mathbf{Alg}(G)\colon \mathbf{Alg}(S) \to \mathbf{Alg}(S)$ by

$$(S(Y) \xrightarrow{b} Y) \longmapsto (TG(Y) \xrightarrow{\beta_Y} GS(Y) \xrightarrow{G(b)} G(Y)).$$

Now we have to establish the bijective correspondence for $\mathbf{Alg}(F) \dashv \mathbf{Alg}(G)$. It takes the form

$$\mathbf{Alg}(F)\begin{pmatrix} T(X) \\ {\scriptstyle a}\downarrow \\ X \end{pmatrix} = \frac{\begin{pmatrix} SF(X) \\ {\scriptstyle \downarrow F(a)\circ\alpha} \\ F(X) \end{pmatrix} \xrightarrow{\quad f \quad} \begin{pmatrix} S(Y) \\ {\scriptstyle \downarrow b} \\ Y \end{pmatrix}}{\begin{pmatrix} T(X) \\ {\scriptstyle a}\downarrow \\ X \end{pmatrix} \xrightarrow{\quad g \quad} \begin{pmatrix} TGY \\ {\scriptstyle \downarrow G(b)\circ\beta} \\ GY \end{pmatrix} = \mathbf{Alg}(G)\begin{pmatrix} S(Y) \\ {\scriptstyle \downarrow b} \\ Y \end{pmatrix}}.$$

This correspondence is the one of the adjunction $F \dashv G$. We need only to check that the transposes are again homomorphisms of algebras. We shall do so for the map f above the lines, and leave the other case (for g) to the interested reader.

Assume therefore that f is a homomorphism of S-algebras as indicated above the lines. This means that $b \circ Sf = f \circ F(a) \circ \alpha_X$. The transpose $\overline{f} = G(f) \circ \eta_X\colon X \to G(Y)$ is then a homomorphism of T-algebras:

$$\begin{aligned}
&(G(b) \circ \beta_Y) \circ T(\overline{f}) \\
&= G(b) \circ GS(\varepsilon_Y) \circ G(\alpha^{-1}_{G(Y)}) \circ \eta_{TG(Y)} \circ TG(f) \circ T(\eta_X) \\
&= G(b) \circ GS(\varepsilon_Y) \circ G(\alpha^{-1}_{G(Y)}) \circ GFTG(f) \circ GFT(\eta_X) \circ \eta_{T(X)} \\
&\qquad \text{by naturality of } \eta \\
&= G(b) \circ GS(\varepsilon_Y) \circ GSFG(f) \circ GSF(\eta_X) \circ G(\alpha^{-1}_X) \circ \eta_{T(X)} \\
&\qquad \text{by naturality of } \alpha \text{ (and thus also of } \alpha^{-1}) \\
&= G(b) \circ GS(f) \circ GS(\varepsilon_{F(X)}) \circ GSF(\eta_X) \circ G(\alpha^{-1}_X) \circ \eta_{T(X)} \\
&\qquad \text{by naturality of } \varepsilon \\
&= G(b) \circ GS(f) \circ G(\alpha^{-1}_X) \circ \eta_{T(X)} \\
&\qquad \text{by one of the triangular identities; see Exercise 2.5.7} \\
&= G(f) \circ GF(a) \circ G(\alpha_X) \circ G(\alpha^{-1}_X) \circ \eta_{T(X)} \\
&\qquad \text{by assumption about } f \\
&= G(f) \circ \eta_X \circ a \\
&\qquad \text{by naturality of } \eta \\
&= \overline{f} \circ a.
\end{aligned}$$

❑

Those readers in search of even more abstraction may want to check that the above pair of functors (S, T) with the natural transformations (α, β) forms a 'map of adjunctions' $(F \dashv G) \to (F \dashv G)$; see [344, IV, 7] for the definition of this notion. The construction $\mathbf{Alg}(F) \dashv \mathbf{Alg}(G)$ may then be understood as algebra of this endomorphism (of adjunctions), in a suitably abstract sense.

This section concludes with a relatively long series of exercises, mostly because adjunctions offer a perspective that leads to many more results. In a particular situation they can concisely capture the essentials of back-and-forth translations.

Exercises

2.5.1 Recall from Exercise 1.4.4 that the assignment $A \mapsto A^\star$ gives a functor $(-)^\star \colon \mathbf{Sets} \to \mathbf{Mon}$ from sets to monoids. Show that this functor is left adjoint to the forgetful functor $\mathbf{Mon} \to \mathbf{Sets}$ which maps a monoid $(M, +, 0)$ to its underlying set M and forgets the monoid structure.

2.5.2 Check that the bijective correspondences

$$\dfrac{\dfrac{X \longrightarrow \mathcal{P}(Y)}{\bullet \subseteq X \times Y}}{\dfrac{\bullet \subseteq Y \times X}{Y \longrightarrow \mathcal{P}(X)}}$$

induced by the correspondence (2.16) give rise to an adjunction of the form $\mathbf{Sets}^{\mathrm{op}} \leftrightarrows \mathbf{Sets}$.

2.5.3 The intention of this exercise is to show that the mapping $\# \mapsto F_\#$ from arities to functors is functorial. This requires some new notation and terminology. Let Endo(**Sets**) be the category with endofunctors $F \colon \mathbf{Sets} \to \mathbf{Sets}$ as objects and natural transformations between them as morphisms. One writes $\mathbf{Sets}/\mathbb{N}$ for the slice category over $\mathbb{N}$; see Exercise 1.4.3.

Prove that mapping $\# \mapsto F_\#$ from (2.18) yields a functor $\mathbf{Sets}/\mathbb{N} \to$ Endo(**Sets**). (This functor $\mathbf{Sets}/\mathbb{N} \to$ Endo(**Sets**) restricts to a 'full and faithful' functor via a suitable restriction of the category Endo(**Sets**); see [4, theorem 3.4]. This means that morphisms $F_{\#_1} \to F_{\#_2}$ in this restricted category are in one-to-one correspondence with morphisms of arities $\#_1 \to \#_2$.)

2.5.4 This exercise describes 'strength' for endofunctors on **Sets**. In general, this is a useful notion in the theory of datatypes [104, 105] and of computations [357]; see Section 5.2 for a systematic description.

Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be an arbitrary functor. Consider for sets X, Y the strength map

$$\begin{array}{rcl} F(X) \times Y & \xrightarrow{\mathrm{st}_{X,Y}} & F(X \times Y) \\ (u, y) & \longmapsto & F(\lambda x \in X.\, (x, y))(u). \end{array} \tag{2.32}$$

1. Prove that this yields a natural transformation $F(-) \times (-) \overset{\mathrm{st}}{\Longrightarrow} F((-) \times (-))$, where both the domain and codomain are functors $\mathbf{Sets} \times \mathbf{Sets} \to \mathbf{Sets}$.
2. Describe this strength map for the list functor $(-)^\star$ and for the powerset functor $\mathcal{P}$.

2.5.5 The following strengthening of induction is sometimes called 'induction with parameters'. It is different from recursion, which also involves an additional parameter; see Proposition 2.4.7.

Assume a functor F with a strength natural transformation as in the previous exercise and with initial algebra $\alpha\colon F(A) \xrightarrow{\cong} A$. Let P be a set (or object) for parameters. Prove that for each map $h\colon F(X) \times P \to X$ there is a unique $f\colon A \times P \to X$ making the following diagram commute:

$$\begin{array}{ccc} F(A) \times P & \xrightarrow{\langle F(f) \circ \mathrm{st}, \pi_2 \rangle} & F(X) \times P \\ {\scriptstyle \alpha \times \mathrm{id}} \downarrow \cong & & \downarrow {\scriptstyle h} \\ A \times P & \xrightarrow[f]{} & X \end{array}$$

Hint: First turn h into a suitable algebra $h'\colon F(X^P) \to X^P$ via strength.

Use this mechanism to define the append map $\mathsf{app}\colon A^\star \times A^\star \to A^\star$ from Example 2.4.6.

2.5.6 Show that the forgetful functor $U\colon \mathbf{Sp} \to \mathbf{Sets}$ from topological spaces to sets has both a left adjoint (via the discrete topology on a set, in which every subset is open) and a right adjoint (via the indiscrete topology, with only the empty set and the whole set itself as open sets).

2.5.7 Assume two functors $F\colon \mathbb{C} \to \mathbb{D}$ and $G\colon \mathbb{D} \to \mathbb{C}$ in opposite directions, with natural transformations $\eta\colon \mathrm{id}_{\mathbb{C}} \Rightarrow GF$ and $\varepsilon\colon FG \Rightarrow$

$\mathrm{id}_{\mathbb{D}}$. Define functions $\psi\colon \mathbb{D}(F(X), Y) \longrightarrow \mathbb{C}(X, G(Y)$ by $\psi(f) = G(f) \circ \eta_X$.

1. Check that such ψs are natural.
2. Prove that these ψs are isomorphisms if and only if the following **triangular identities** hold:

$$G(\varepsilon) \circ \eta G = \mathrm{id}, \qquad \varepsilon F \circ F(\eta) = \mathrm{id}.$$

2.5.8 A morphism $m\colon X' \to X$ in a category $\mathbb{D}$ is called a **monomorphism** (or **mono**, for short), written as $m\colon X' \rightarrowtail X$, if for each parallel pair of arrows $f, g\colon Y \to X'$, $m \circ f = m \circ g$ implies $f = g$.

1. Prove that the monomorphisms in **Sets** are precisely the injective functions.
2. Let $G\colon \mathbb{D} \to \mathbb{C}$ be a right adjoint. Show that if m is a monomorphism in $\mathbb{D}$, then so is $G(m)$ in $\mathbb{C}$.

Dually, an **epimorphism** (or **epi**, for short) in $\mathbb{C}$ is an arrow written as $e\colon X' \twoheadrightarrow X$ such that for all maps $f, g\colon X \to Y$, if $e \circ f = e \circ g$, then $f = g$.

3. Show that the epimorphism in **Sets** are the surjective functions. *Hint*: For an epi $X \twoheadrightarrow Y$, choose two appropriate maps $Y \to 1 + Y$.
4. Prove that left adjoints preserve epimorphism.

2.5.9 Notice that the existence of final coalgebras for finite polynomial functors (Theorem 2.3.9) that is used in the proof of Proposition 2.5.3 is actually a special case of this proposition. *Hint*: Consider the right adjoint at the final set 1.

2.5.10 Assume an endofunctor $F\colon \mathbb{C} \to \mathbb{C}$. Prove that there are natural transformations

$$\mathbf{CoAlg}(F) \underset{FU}{\overset{U}{\Downarrow}} \mathbb{C} \qquad \text{and} \qquad \mathbf{Alg}(F) \underset{U}{\overset{FU}{\Downarrow}} \mathbb{C}$$

where U is the forgetful functor. (In 2-categorical terminology these maps form inserters; see e.g. [218, appendix].)

2.5.11 (Hughes) Let $\mathbb{C}$ be an arbitrary category with products $\times$ and let $F, H\colon \mathbb{C} \to \mathbb{C}$ be two endofunctors on $\mathbb{C}$. Assume that cofree F-coalgebras exist, i.e. that the forgetful functor $U\colon \mathbf{CoAlg}(F) \to \mathbb{C}$ has a right adjoint G – as in Proposition 2.5.3. Prove then that there is an isomorphism of categories of coalgebras,

$$\mathbf{CoAlg}(F \times H) \xrightarrow{\cong} \mathbf{CoAlg}(GHU),$$

where $\mathbf{CoAlg}(GHU)$ is a category of coalgebras on coalgebras, for the functor composition $GHU\colon \mathbf{CoAlg}(F) \to \mathbb{C} \to \mathbb{C} \to \mathbf{CoAlg}(F)$.

2.5.12 Consider two adjoint endofunctors as in

$$F \circlearrowright \mathbb{C} \circlearrowleft G \qquad \text{with} \qquad F \dashv G.$$

Prove that we then get an isomorphism of categories

$$\mathbf{Alg}(F) \xrightarrow{\cong} \mathbf{CoAlg}(G)$$

between the associated categories of algebras and coalgebras.
(Remark: As noted in [36, theorem 5.7], when $\mathbb{C} = \mathbf{Sets}$ the only such adjunctions $F \dashv G$ are product-exponent adjunctions $X \times (-) \dashv (-)^X$ as in (2.11). The argument goes as follows. In **Sets**, each object A can be written as coproduct $\coprod_{a \in A} 1$ of singletons. A left adjoint F must preserve such coproducts, so that $F(A) \cong \coprod_{a \in A} F(1) \cong F(1) \times A$. But then $G(-) \cong F(1) \Rightarrow (-)$, by uniqueness of adjoints.)

2.5.13 Theorem 2.5.9 deals with lifting adjunctions to categories of algebras. Check that its 'dual' version for coalgebras is (see [218]) the following:

For functors $T\colon \mathbb{B} \to \mathbb{B}$, $G\colon \mathbb{B} \to \mathbb{A}$, $S\colon \mathbb{A} \to \mathbb{A}$, a natural transformation $\alpha\colon GS \Rightarrow TG$ induces a functor $\mathbf{CoAlg}(G)\colon \mathbf{CoAlg}(S) \to \mathbf{CoAlg}(T)$. Furthermore, if α is an isomorphism, then a left adjoint $F \dashv G$ induces a left adjoint $\mathbf{CoAlg}(F) \dashv \mathbf{CoAlg}(G)$. Does it require a new proof?

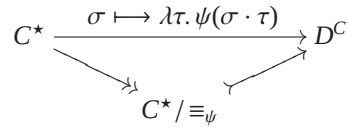
2.5.14 A deterministic automaton $\langle \delta, \epsilon \rangle\colon X \to X^A \times B$ is called **observable** if its behaviour function $\mathrm{beh} = \lambda x.\, \lambda \sigma.\, \epsilon(\delta^*(x, \sigma))\colon X \to B^{A^\star}$ from Proposition 2.3.5 is injective. Later, in Corollary 3.4.3 we shall see that this means that bisimilar states are equal.

If this automaton comes equipped with an initial state $x_0 \in X$ one calls the automaton **reachable** if the function $\delta^*(x_0, -)\colon A^\star \to X$ from Section 2.2 is surjective. This means that every state can be reached from the initial state x_0 via a suitable sequence of inputs. The automaton is called **minimal** if it is both observable and reachable.

The realisation construction $\mathcal{R}\colon \mathbf{DB} \to \mathbf{DA}$ from Proposition 2.5.8 clearly yields an observable automaton since the resulting behaviour function is the identity. An alternative construction, the **Nerode realisation**, gives a minimal automaton. It is obtained from a behaviour $\psi\colon C^\star \to D$ as follows. Consider the equivalence relation $\equiv_\psi \subseteq C^\star \times C^\star$ defined by

$$\sigma \equiv_\psi \sigma' \iff \forall \tau \in C^\star . \psi(\sigma \cdot \tau) = \psi(\sigma' \cdot \tau).$$

We take the quotient $C^\star / \equiv_\psi$ as state space; it is defined as factorisation:



This quotient carries an automaton structure with transition function $\delta_\psi : (C^\star / \equiv_\psi) \to (C^\star / \equiv_\psi)^C$ given by $\delta_\psi([\sigma])(c) = [\sigma \cdot c]$, observation function $\epsilon_\psi : (C^\star / \equiv_\psi) \to D$ defined as $\epsilon_\psi([\sigma]) = \psi(\sigma)$, and initial state $[\langle\rangle] \in C^\star / \equiv_\psi$.

1. Check that this Nerode realisation $\mathcal{N}(C^\star \xrightarrow{\psi} D)$ is indeed a minimal automaton, forming a subautomaton/subcoalgebra $C^\star / \equiv_\psi \rightarrowtail D^{C^\star}$ of the final coalgebra.

Write **RDA** for the 'subcategory' of **DA** with reachable automata as objects, and morphisms (f, g, h) as in **DA** but with f a *surjective* function between the input sets. Similarly, let **RDB** be the subcategory of **DB** with the same objects but with morphisms (f, g) where f is surjective.

2. Check that the behaviour functor $\mathcal{B} : \mathbf{DA} \to \mathbf{DB}$ from Proposition 2.5.8 restricts to a functor $\mathcal{B} : \mathbf{RDA} \to \mathbf{RDB}$, and show that it has Nerode realisation $\mathcal{N}$ yields a functor $\mathbf{RDA} \to \mathbf{RDB}$ in the opposite direction.
3. Prove that there is an adjunction $\mathcal{B} \dashv \mathcal{N}$.
4. Let **MDA** be the 'subcategory' of **RDA** with minimal automata as objects. Check that the adjunction in the previous point restricts to an equivalence of categories $\mathbf{MDA} \simeq \mathbf{RDB}$. Thus, (states of) minimal automata are in fact (elements of) final coalgebras. (This result comes from [161, 162]; see also [14].)

2.5.15 This exercise and the next one continue the description of linear dynamical systems from Exercise 2.2.12. Here we look at the duality between reachability and observability. First a preliminary result.

1. Let B be an arbitrary vector space. Prove that the final coalgebra in **Vect** of the functor $X \mapsto B \times X$ is the set of infinite sequences $B^{\mathbb{N}}$, with obvious vector space structure, and with coalgebra structure $\langle \mathsf{hd}, \mathsf{tl} \rangle : B^{\mathbb{N}} \xrightarrow{\cong} B \times B^{\mathbb{N}}$ given by head and tail.

Call a linear dynamical system $A \xrightarrow{F} X \xrightarrow{G} X \xrightarrow{H} B$ **reachable** if the induced function $\mathrm{int}_{[F,G]} \colon A^{\S} \to X$ in the diagram on the left below is surjective (or equivalently, an epimorphism in **Vect**).

$$\begin{array}{ccc} A + A^{\S} & \xrightarrow{\mathrm{id}_A + \mathrm{int}_{[F,G]}} & A + X \\ {\scriptstyle [\mathsf{in},\mathsf{sh}]}\downarrow & \cong & \downarrow{\scriptstyle [F,G]} \\ A^{\S} & \xrightarrow[\mathrm{int}_{[F,G]}]{} & X \end{array} \qquad \begin{array}{ccc} B \times X & \xrightarrow{\mathrm{id}_B \times \mathrm{beh}_{\langle H,G\rangle}} & B \times B^{\mathbb{N}} \\ \uparrow{\scriptstyle \langle H,G\rangle} & & \cong \uparrow{\scriptstyle \langle \mathsf{hd},\mathsf{tl}\rangle} \\ X & \xrightarrow[\mathrm{beh}_{\langle H,G\rangle}]{} & B^{\mathbb{N}} \end{array}$$

Similarly, call this system **observable** if the map $\mathrm{beh}_{\langle H,G\rangle} \colon X \to B^{\mathbb{N}}$ on the right is injective (equivalently, a monomorphism in **Vect**). And call the system **minimal** if it is both reachable and observable.

2. Prove that $A \xrightarrow{F} X \xrightarrow{G} X \xrightarrow{H} B$ is reachable in **Vect** if and only if $B \xrightarrow{H} X \xrightarrow{G} X \xrightarrow{F} A$ is observable in $\mathbf{Vect}^{\mathrm{op}}$.

(Kalman's duality result [287, chapter 2] is now an easy consequence in *finite-dimensional* vector spaces, where the adjoint operator $(-)^*$ makes **Vect** isomorphic to $\mathbf{Vect}^{\mathrm{op}}$ – where V^* is of course the 'dual' vector space of linear maps to the underlying field. This result says that $A \xrightarrow{F} X \xrightarrow{G} X \xrightarrow{H} B$ is reachable if and only if $B^* \xrightarrow{H^*} X^* \xrightarrow{G^*} X^* \xrightarrow{F^*} A^*$ is observable. See also [36]. There is also a duality result for bialgebras in [69].)

2.5.16 This exercise sketches an adjunction capturing Kalman's minimal realisation [287, chapter 10] for linear dynamical systems, in analogy with the Nerode realisation, described in Exercise 2.5.14. This categorical version is based on [35, 36].

Form the category **RLDS** of reachable linear dynamical systems. Its objects are such reachable systems $A \xrightarrow{F} X \xrightarrow{G} X \xrightarrow{H} B$ in **Vect**. And its morphisms from $A \xrightarrow{F} X \xrightarrow{G} X \xrightarrow{H} B$ to $C \xrightarrow{F'} Y \xrightarrow{G'} Y \xrightarrow{H'} D$ are triples of functions $C \overset{f}{\twoheadrightarrow} A$, $B \xrightarrow{g} D$ and $X \xrightarrow{h} Y$ with

$$\begin{array}{ccccccc} A & \xrightarrow{F} & X & \xrightarrow{G} & X & \xrightarrow{H} & B \\ {\scriptstyle f}\uparrow\!\!\uparrow & & \downarrow{\scriptstyle h} & & \downarrow{\scriptstyle h} & & \downarrow{\scriptstyle g} \\ C & \xrightarrow[F']{} & Y & \xrightarrow[G']{} & Y & \xrightarrow[H']{} & D \end{array}$$

Note that f is required to be a surjection/epimorphism.

Also, there is a category **RLB** with linear maps $\varphi\colon A^{\S} \to B$ as objects. A morphism $(A^{\S} \xrightarrow{\varphi} B) \longrightarrow (C^{\S} \xrightarrow{\psi} D)$ is a pair of linear maps $f\colon C \twoheadrightarrow A$ and $g\colon B \to D$ with $g \circ \varphi \circ f^{\S} = \psi$ – where $f^{\S}$ results from the functoriality of $(-)^{\S}$; see Exercise 2.4.8.4.

1. Demonstrate that the behaviour formula from Exercises 2.2.12.4 and 2.4.8.3 yields a behaviour functor $\mathcal{B}\colon$ **RLDS** $\to$ **RLB**, given by $(F, G, H) \longmapsto H \circ \mathrm{int}_{[F,G]}$.
2. Construct a functor $\mathcal{K}\colon$ **RLB** $\to$ **RLDS** in the reverse direction in the following way. Assume a behaviour $\psi\colon C^{\S} \to D$, and form the behaviour map $b = \mathrm{beh}_{\langle \psi, \mathsf{sh}\rangle}\colon C^{\S} \to D^{\mathbb{N}}$ below, using the finality from the previous exercise:

$$\begin{array}{ccc} D \times C^{\S} & \xrightarrow{\mathrm{id}_D \times b} & D \times D^{\mathbb{N}} \\ {\scriptstyle \langle \psi, \mathsf{sh}\rangle}\uparrow & & \uparrow{\scriptstyle \cong\ \langle \mathsf{hd}, \mathsf{tl}\rangle} \\ C^{\S} & \xrightarrow[b]{} & D^{\mathbb{N}} \end{array}$$

The image $\mathrm{Im}(b)$ of this behaviour map can be written as

$$\Big(C^{\S} \xrightarrow{\mathrm{beh}_{\langle \psi, \mathsf{sh}\rangle}} D^{\mathbb{N}}\Big) = \Big(C^{\S} \xrightarrow{e} \mathrm{Im}(b) \xrightarrow{m} D^{\mathbb{N}}\Big).$$

It is not hard to see that the tail function $\mathsf{tl}\colon B^{\mathbb{N}} \to B^{\mathbb{N}}$ restricts to $\mathsf{tl}'\colon \mathrm{Im}(b) \to \mathrm{Im}(b)$ via diagonal fill-in:

$$\begin{array}{ccc} C^{\S} & \xrightarrow{e} & \mathrm{Im}(b) \\ {\scriptstyle e \circ \mathsf{sh}}\downarrow & \swarrow{\scriptstyle \mathsf{tl}'} & \downarrow{\scriptstyle \mathsf{tl} \circ m} \\ \mathrm{Im}(b) & \xrightarrow[m]{} & D^{\mathbb{N}} \end{array}$$

Hence one can define a linear dynamical system as

$$\mathcal{K}\Big(C^{\S} \xrightarrow{\psi} D\Big) \stackrel{\text{def}}{=} \Big(C \xrightarrow{e \circ \mathsf{in}} \mathrm{Im}(b) \xrightarrow{\mathsf{tl}'} \mathrm{Im}(b) \xrightarrow{\mathsf{hd} \circ m} D\Big).$$

Prove that this gives a minimal realisation in an adjunction $\mathcal{B} \dashv \mathcal{K}$.

2.5.17 This exercise describes the so-called terms-as-natural-transformations view which originally stems from [329]. It is elaborated in a coalgebraic context in [325].

Let $H\colon$ **Sets** $\to$ **Sets** be a (not necessarily polynomial) functor, with free algebras, given by a left adjoint F to the forgetful functor

$U\colon \mathbf{Alg}(H) \to \mathbf{Sets}$. Let X be an arbitrary set whose elements are considered as variables. Elements of the carrier $UF(X)$ of the free algebra on X can then be seen as terms containing free variables from X. Show that there is a bijective correspondence:

$$\frac{\text{terms } t \in UF(X)}{\text{natural transformations } \tau\colon U^X \Longrightarrow U}\,.$$

Hint: The component of the natural transformation at a specific algebra $HA \to A$ is the mapping which takes a valuation $\rho\colon X \to A$ of the variables in A to an interpretation $[\![\, t \,]\!]^A_\rho$ of the term t in the algebra A. Naturality then says that for a homomorphism $f\colon A \to B$ of algebras, one has the familiar equation $f([\![\, t \,]\!]^A_\rho) = [\![\, t \,]\!]^B_{f\circ\rho}$.