1
Motivation

This chapter tries to explain why coalgebras are interesting structures in mathematics and computer science. It does so via several examples. The notation used for these examples will be explained informally, as we proceed. The emphasis at this stage is not so much on precision in explanation but on transfer of ideas and intuitions. Therefore, for the time being we define a coalgebra – very informally – to be a function of the form

$$S \xrightarrow{\quad c \quad} \boxed{\;\cdots \quad S \quad \cdots\;} \,. \tag{1.1}$$

What we mean is: a coalgebra is given by a set S and a function c with S as domain and with a 'structured' codomain (result, output, the box $\boxed{\cdots}$), in which the domain S may occur again. The precise general form of these codomain boxes is not of immediate concern.

Some terminology: We often call S the *state space* or *set of states* and say that the coalgebra *acts on* S. The function c is sometimes called the *transition function* or *transition structure*. The idea that will be developed is that coalgebras describe general 'state-based systems' provided with 'dynamics' given by the function c. For a state $x \in S$, the result $c(x)$ tells us what the successor states of x are, if any. The codomain $\boxed{\cdots}$ is often called the *type* or *interface* of the coalgebra. Later we shall see that it is a *functor*.

A simple example of a coalgebra is the function

$$\mathbb{Z} \xrightarrow{\quad n \,\mapsto\, (n-1,\, n+1) \quad} \mathbb{Z} \times \mathbb{Z}$$

with state space $\mathbb{Z}$ occurring twice on the right-hand side. Thus the box or type of this coalgebra is: $\boxed{(-) \times (-)}$. The transition function $n \mapsto (n-1, n+1)$

may also be written using λ-notation as $\lambda n.\,(n-1, n+1)$ or as $\lambda n \in \mathbb{Z}.\,(n-1, n+1)$.

Another example of a coalgebra, this time with state space the set $A^{\mathbb{N}}$ of functions from $\mathbb{N}$ to some given set A, is

$$A^{\mathbb{N}} \xrightarrow{\sigma \mapsto (\sigma(0),\, \lambda n.\, \sigma(n+1))} A \times A^{\mathbb{N}}.$$

In this case the box is $\boxed{A \times (-)}$. If we write σ as an infinite sequence $(\sigma_n)_{n\in\mathbb{N}}$ we may write this coalgebra as a pair of functions $\langle \mathsf{head}, \mathsf{tail}\rangle$ where

$$\mathsf{head}((\sigma_n)_{n\in\mathbb{N}}) = \sigma_0 \quad \text{and} \quad \mathsf{tail}((\sigma_n)_{n\in\mathbb{N}}) = (\sigma_{n+1})_{n\in\mathbb{N}}.$$

Many more examples of coalgebras will occur throughout this text.

This chapter is devoted to ‘selling’ and ‘promoting’ coalgebras. It does so by focusing on the following topics.

1. A representation as a coalgebra (1.1) is often very natural, from the perspective of state-based computation.
2. There are powerful ‘coinductive’ definition and proof principles for coalgebras.
3. There is a very natural (and general) temporal logic associated with coalgebras.
4. The coalgebraic notions are on a suitable level of abstraction, so that they can be recognised and used in various settings.

Full appreciation of this last point requires some familiarity with basic category theory. It will be provided in Section 1.4.

Remark 1.0.1 Readers with a mathematical background may be familiar with the notion of coalgebra as comonoid in vector spaces, dual to an algebra as a monoid. In that case one has a ‘counit’ map $V \to K$, from the carrier space V to the underlying field K, together with a ‘comultiplication’ $V \to V \otimes V$. These two maps can be combined into a single map $V \to K \times (V \otimes V)$ of the form (1.1), forming a coalgebra in the present sense. The notion of coalgebra used here is thus much more general than the purely mathematical one.

1.1 Naturalness of Coalgebraic Representations

We turn first to an area where coalgebraic representations as in (1.1) occur naturally and may be useful, namely programming languages – used for writing computer programs. What are programs, and what do they do? Well, programs are lists of instructions telling a computer what to do. Fair enough. But what are programs from a mathematical point of view? Put differently, what do programs mean?[1] One view is that programs are certain functions that take an input and use it to compute a certain result. This view does not cover all programs: certain programs, often called processes, are meant to be running forever, like operating systems, without really producing a result. But we shall follow the view of programs as functions for now. The programs we have in mind work not only on input, but also on what is usually called a state, for example for storing intermediate results. The effect of a program on a state is not immediately visible and is therefore often called the *side effect* of the program. One may think of the state as given by the contents of the memory in the computer that is executing the program. This is not directly observable.

Our programs should thus be able to modify a state, typically via an assignment like `i = 5` in a so-called imperative programming language. Such an assignment statement is interpreted as a function that turns a state x into a new, successor state x' in which the value of the identifier `i` is equal to `5`. Statements in such languages are thus described via suitable 'state transformer' functions. In simplest form, ignoring input and output, they map a state to a successor state, as in

$$S \xrightarrow{\;\mathsf{stat}\;} S, \tag{1.2}$$

where we have written S for the set of states. Its precise structure is not relevant. Often the set S of states is considered to be a 'black box' to which we do not have direct access, so that we can observe only certain aspects. For instance, the function $i\colon S \to \mathbb{Z}$ representing the above integer `i` allows us to observe the value of i. The value $i(x')$ should be 5 in the result state x' after evaluating the assignment `i = 5`, considered as a function $S \to S$, as in (1.2).

This description of statements as functions $S \to S$ is fine as first approximation, but one quickly realises that statements do not always terminate normally and produce a successor state. Sometimes they can 'hang' and continue to compute without ever producing a successor state. This typically happens because of an infinite loop, for example in a `while` statement or because of a recursive call without exit.

There are two obvious ways to incorporate such non-termination.

1. **Adjust the state space**. In this case one extends the state space S to a space $S_\perp \overset{\text{def}}{=} \{\perp\} \cup S$, where $\perp$ is a new 'bottom' element not occurring in

[1] This question comes up frequently when confronted with two programs – one possibly as a transformation from the other – which perform the same task in a different manner and which could thus be seen as the same program. But how can one make precise that they are the same?

S that is especially used to signal non-termination. Statements then become functions:

$$S_\perp \xrightarrow{\ \mathsf{stat}\ } S_\perp \quad \text{with the requirement} \quad \mathsf{stat}(\perp) = \perp.$$

The side-condition expresses the idea that once a statement hangs it will continue to hang.

The disadvantage of this approach is that the state space becomes more complicated and that we have to make sure that all statements satisfy the side-condition, namely that they preserve the bottom element $\perp$. But the advantage is that composition of statements is just function composition.

2. **Adjust the codomain**. The second approach keeps the state space S as it is but adapts the codomain of statements, as in

$$S \xrightarrow{\ \mathsf{stat}\ } S_\perp \quad \text{where, recall,} \quad S_\perp = \{\perp\} \cup S.$$

In this representation we easily see that in each state $x \in S$ the statement can either hang, when $\mathsf{stat}(x) = \perp$, or terminate normally, namely when $\mathsf{stat}(x) = x'$ for some successor state $x' \in S$. What is good is that there are no side-conditions anymore. But composition of statements cannot be defined via function composition, because the types do not match. Thus the types force us to deal explicitly with the propagation of non-termination: for these kind of statements $s_1, s_2 \colon S \to S_\perp$ the composition $s_1 ; s_2$, as a function $S \to S_\perp$, is defined via a case distinction (or pattern match) as

$$s_1 ; s_2 = \lambda x \in S.\begin{cases} \perp & \text{if } s_1(x) = \perp \\ s_2(x') & \text{if } s_1(x) = x'. \end{cases}$$

This definition is more difficult than function composition (as used in point 1 above), but it explicitly deals with the case distinction that is of interest, namely between non-termination and normal termination. Hence being forced to make these distinctions explicitly is maybe not so bad at all.

We push these same ideas a bit further. In many programming languages (such as Java [43]) programs may not only hang, but also terminate ‘abruptly’ because of an exception. An exception arises when some constraint is violated, such as a division by zero or an access `a[i]` in an array `a` which is a null-reference. Abrupt termination is fundamentally different from non-termination: non-termination is definitive and irrevocable, whereas a program can recover from abrupt termination via a suitable exception handler that restores normal termination. In Java this is done via a `try-catch` statement; see for instance [43, 173, 240].

Let us write E for the set of exceptions that can be thrown. Then there are again two obvious representations of statements that can terminate normally or abruptly or can hang.

1. **Adjust the state space**. Statements then remain endofunctions[2] on an extended state space:

$$\big(\{\perp\} \cup S \cup (S \times E)\big) \xrightarrow{\quad \mathsf{stat} \quad} \big(\{\perp\} \cup S \cup (S \times E)\big).$$

   The entire state space clearly becomes complicated now. But also the side-conditions are becoming non-trivial: we still want $\mathsf{stat}(\perp) = \perp$, and also $\mathsf{stat}(x, e) = (x, e)$, for $x \in S$ and $e \in E$, but the latter only for non-catch statements. Keeping track of such side-conditions may easily lead to mistakes. But on the positive side, composition of statements is still function composition in this representation.

2. **Adjust the codomain**. The alternative approach is again to keep the state space S as it is, but to adapt the codomain type of statements, namely as

$$S \xrightarrow{\quad \mathsf{stat} \quad} \big(\{\perp\} \cup S \cup (S \times E)\big). \tag{1.3}$$

   Now we do not have side-conditions and we can clearly distinguish the three possible termination modes of statements. This structured output type in fact forces us to make these distinctions in the definition of the composition $s_1 \,;\, s_2$ of two such statements $s_1, s_2 \colon S \to \{\perp\} \cup S \cup (S \times E)$, as in

$$s_1 \,;\, s_2 \;=\; \lambda x \in S.\, \begin{cases} \perp & \text{if } s_1(x) = \perp \\ s_2(x') & \text{if } s_1(x) = x' \\ (x', e) & \text{if } s_1(x) = (x', e). \end{cases}$$

   Thus, if s_1 hangs or terminates abruptly, then the subsequent statement s_2 is not executed. This is very clear in this second *coalgebraic* representation.

   When such a coalgebraic representation is formalised within the typed language of a theorem prover (as in [265]), the type checker of the theorem prover will make sure that appropriate case distinctions are made, according to the output type as in (1.3). See also [240] where Java's exception mechanism is described via such case distinctions, closely following the official language definition [173].

These examples illustrate that coalgebras as functions with structured codomains $\boxed{\cdots}$, as in (1.1), arise naturally and that the structure of the

2 An endofunction is a function $A \to A$ from a set A to itself.

codomain indicates the kind of computations that can be performed. This idea will be developed further and applied to various forms of computation. For instance, non-deterministic statements may be represented via the powerset $\mathcal{P}$ as coalgebraic state transformers $S \to \mathcal{P}(S)$ with multiple result states. But there are many more such examples, involving for instance probability distributions on states.

(Readers familiar with computational monads [357] may recognise similarities. Indeed, in a computational setting there is a close connection between coalgebraic and monadic representations. Briefly, the monad introduces the computational structure, like composition and extension, whereas the coalgebraic view leads to an appropriate program logic. This is elaborated for Java in [264].)

Exercises

1.1.1 1. Prove that the composition operation ; as defined for coalgebras $S \to \{\bot\} \cup S$ is associative, i.e. satisfies $s_1 ;(s_2 ; s_3) = (s_1 ; s_2) ; s_3$, for all statements $s_1, s_2, s_3 \colon S \to \{\bot\} \cup S$.

Define a statement $\mathsf{skip} \colon S \to \{\bot\} \cup S$ which is a unit for composition ; i.e. which satisfies $(\mathsf{skip} ; s) = s = (s ; \mathsf{skip})$, for all $s \colon S \to \{\bot\} \cup S$.

2. Do the same for ; defined on coalgebras $S \to \{\bot\} \cup S \cup (S \times E)$.

(In both cases, statements with an associative composition operation and a unit element form a monoid.)

1.1.2 Define also a composition monoid $(\mathsf{skip}, ;)$ for coalgebras $S \to \mathcal{P}(S)$.

1.2 The Power of the Coinduction

In this section we shall look at sequences – or lists or words, as they are also called. Sequences are basic data structures, both in mathematics and in computer science. One can distinguish finite sequences $\langle a_1, \ldots, a_n \rangle$ and infinite $\langle a_1, a_2, \ldots \rangle$ ones. The mathematical theory of finite sequences is well understood and a fundamental part of computer science, used in many programs (notably in the language Lisp). Definition and reasoning with finite lists is commonly done with induction. As we shall see, infinite lists require *coinduction*. Infinite sequences can arise in computing as the observable outcomes of a program that runs forever. Also, in functional programming, they can occur as so-called lazy lists, as in the languages Haskell [71] or Clean [381]. Modern extensions of logical programming languages have support for infinite sequences [431, 189]. (Logic programming languages themselves can also be described coalgebraically; see [78, 107, 305, 306, 80, 81].)

In the remainder of this section we shall use an arbitrary but fixed set A and wish to look at both finite $\langle a_1, \ldots, a_n \rangle$ and infinite $\langle a_1, a_2, \ldots \rangle$ sequences of elements a_i of A. The set A may be understood as a parameter, and our sequences are thus parametrised by A, or, put differently, are polymorphic in A.

We shall develop a slightly unusual and abstract perspective on sequences. It treats sequences not as completely given at once but as arising in a local, step-by-step manner. This coalgebraic approach relies on the following basic fact. It turns out that the set of both finite and infinite sequences enjoys a certain 'universal' property, namely that it is a *final* coalgebra (of suitable type). We shall explain what this means and how this special property can be exploited to define various operations on sequences and to prove properties about them. A special feature of this universality of the final coalgebra of sequences is that it avoids making the (global) distinction between finiteness and infiniteness for sequences.

First some notation. We write $A^\star$ for the set of *finite* sequences $\langle a_1, \ldots, a_n \rangle$ (or lists or words) of elements $a_i \in A$, and $A^\mathbb{N}$ for the set of infinite ones: $\langle a_1, a_2, \ldots \rangle$. The latter may also be described as functions $a_{(-)} \colon \mathbb{N} \to A$, which explains the exponent notation in $A^\mathbb{N}$. Sometimes, the infinite sequences in $A^\mathbb{N}$ are called *streams*. Finally, the set of both finite and infinite sequences A^∞ is then the (disjoint) union $A^\star \cup A^\mathbb{N}$.

The set of sequences A^∞ carries a coalgebra or transition structure, which we simply call next. It tries to decompose a sequence into its head and tail, if any. Hence one may understand next as a partial function. But we describe it as a total function which possibly outputs a special element $\perp$ for undefined:

$$\begin{array}{rcl} A^\infty & \xrightarrow{\quad\mathsf{next}\quad} & \{\perp\} \cup (A \times A^\infty) \\ \sigma & \longmapsto & \begin{cases} \perp & \text{if } \sigma \text{ is the empty sequence } \langle\rangle \\ (a, \sigma') & \text{if } \sigma = a \cdot \sigma' \text{ with head } a \in A \text{ and tail } \sigma' \in A^\infty. \end{cases} \end{array} \tag{1.4}$$

The type of the coalgebra is thus $\boxed{\{\perp\} \cup (A \times (-))}$, as in (1.1), with A^∞ as state space that is plugged in the hole $(-)$ in the box. The successor of a state $\sigma \in A^\infty$, if any, is its tail sequence, obtained by removing the head.

The function next captures the external view on sequences: it tells what can be *observed* about a sequence σ, namely whether or not it is empty, and if not, what its head is. By repeated application of the function next all observable elements of the sequence appear. This 'observational' approach is fundamental in coalgebra.

A first point to note is that this function next is an isomorphism: its inverse next^{-1} sends $\perp$ to the empty sequence $\langle\rangle$ and a pair $(a, \tau) \in A \times A^\infty$ to the sequence $a \cdot \tau$ obtained by prefixing a to τ.

The following result describes a crucial 'finality' property of sequences that can be used to characterise the set A^∞. Indeed, as we shall see later in Lemma 2.3.3, final coalgebras are unique, up-to-isomorphism.

Proposition 1.2.1 (Finality of sequences) *The coalgebra $\mathsf{next}\colon A^\infty \to \{\perp\} \cup A \times A^\infty$ from (1.4) is final among coalgebras of this type: for an arbitrary coalgebra $c\colon S \to \{\perp\} \cup (A \times S)$ on a set S there is a unique 'behaviour' function $\mathrm{beh}_c\colon S \to A^\infty$ which is a homomorphism of coalgebras. That is, for each $x \in S$, both*

- *if $c(x) = \perp$, then $\mathsf{next}(\mathrm{beh}_c(x)) = \perp$.*
- *if $c(x) = (a, x')$, then $\mathsf{next}(\mathrm{beh}_c(x)) = (a, \mathrm{beh}_c(x'))$.*

Both these two points can be combined in a commuting diagram, namely as

$$\begin{tikzcd}
\{\perp\} \cup (A \times S) \arrow[rr, dashed, "\mathrm{id} \cup (\mathrm{id} \times \mathrm{beh}_c)"] & & \{\perp\} \cup (A \times A^\infty) \\
S \arrow[u, "c"] \arrow[rr, dashed, "\mathrm{beh}_c"'] & & A^\infty \arrow[u, "\cong", "\mathsf{next}"']
\end{tikzcd}$$

where the function $\mathrm{id} \cup (\mathrm{id} \times \mathrm{beh}_c)$ on top maps $\perp$ to $\perp$ and (a, x) to $(a, \mathrm{beh}_c(x))$.

In general, we shall write dashed arrows, like above, for maps that are uniquely determined. In the course of this chapter we shall see that a general notion of homomorphism between coalgebras (of the same type) can be defined by such commuting diagrams.

Proof The idea is to obtain the required behaviour function $\mathrm{beh}_c\colon S \to A^\infty$ via repeated application of the given coalgebra c as follows.

$$\mathrm{beh}_c(x) = \begin{cases} \langle\rangle & \text{if } c(x) = \perp \\ \langle a\rangle & \text{if } c(x) = (a, x') \wedge c(x') = \perp \\ \langle a, a'\rangle & \text{if } c(x) = (a, x') \wedge c(x') = (a', x'') \wedge c(x'') = \perp \\ \vdots & \end{cases}$$

Doing this formally requires some care. We define for $n \in \mathbb{N}$ an iterated version $c^n\colon S \to \{\perp\} \cup A \times S$ of c as

$$\begin{aligned} c^0(x) &= c(x) \\ c^{n+1}(x) &= \begin{cases} \perp & \text{if } c^n(x) = \perp \\ c(y) & \text{if } c^n(x) = (a, y). \end{cases} \end{aligned}$$

Obviously, $c^n(x) \neq \bot$ implies $c^m(x) \neq \bot$, for $m < n$. Thus we can define

$$\mathrm{beh}_c(x) = \begin{cases} \langle a_0, a_1, a_2, \ldots\rangle & \text{if } \forall n \in \mathbb{N}.\, c^n(x) \neq \bot \text{, and } c^i(x) = (a_i, x_i) \\ \langle a_0, \ldots, a_{m-1}\rangle & \text{if } m \in \mathbb{N} \text{ is the least number with } c^m(x) = \bot, \\ & \text{and } c^i(x) = (a_i, x_i) \text{, for } i < m. \end{cases}$$

We check the two conditions for homomorphism from the proposition above.

- If $c(x) = \bot$, then the least m with $c^m(x) = \bot$ is 0, so that $\mathrm{beh}_c(x) = \langle\rangle$, and thus also $\mathsf{next}(\mathrm{beh}_c(x)) = \bot$.
- If $c(x) = (a, x')$, then we distinguish two cases:
  - If $\forall n \in \mathbb{N}.\, c^n(x) \neq \bot$, then $\forall n \in \mathbb{N}.\, c^n(x') \neq \bot$, and $c^{i+1}(x) = c^i(x')$. Let $c^i(x') = (a_i, x_i)$, then
  $$\begin{aligned} \mathsf{next}(\mathrm{beh}_c(x)) &= \mathsf{next}(\langle a, a_0, a_1, \ldots\rangle) \\ &= (a, \langle a_0, a_1, \ldots\rangle) \\ &= (a, \mathrm{beh}_c(x')). \end{aligned}$$
  - If m is least with $c^m(x) = \bot$, then $m > 0$ and $m - 1$ is the least k with $c^k(x') = \bot$. For $i < m - 1$ we have $c^{i+1}(x) = c^i(x')$, and thus by writing $c^i(x') = (a_i, x_i)$, we get as before:
  $$\begin{aligned} \mathsf{next}(\mathrm{beh}_c(x)) &= \mathsf{next}(\langle a, a_0, a_1, \ldots, a_{m-2}\rangle) \\ &= (a, \langle a_0, a_1, \ldots, a_{m-2}\rangle) \\ &= (a, \mathrm{beh}_c(x')). \end{aligned}$$

Finally, we still need to prove that this behaviour function beh_c is the unique homomorphism from c to next. Thus, assume also $g\colon S \to A^\infty$ is such that $c(x) = \bot \Rightarrow \mathsf{next}(g(x)) = \bot$ and $c(x) = (a, x') \Rightarrow \mathsf{next}(g(x)) = (a, g(x'))$. We then distinguish:

- $g(x)$ is infinite, say $\langle a_0, a_1, \ldots\rangle$. Then one shows by induction that for all $n \in \mathbb{N}$, $c^n(x) = (a_n, x_n)$, for some x_n. This yields $\mathrm{beh}_c(x) = \langle a_0, a_1, \ldots\rangle = g(x)$.
- $g(x)$ is finite, say $\langle a_0, \ldots, a_{m-1}\rangle$. Then one proves that for all $n < m$, $c^n(x) = (a_n, x_n)$, for some x_n, and $c^m(x) = \bot$. So also now, $\mathrm{beh}_c(x) = \langle a_0, \ldots, a_{m-1}\rangle = g(x)$. ❑

Before exploiting this finality result we illustrate the behaviour function.

Example 1.2.2 (Decimal representations as behaviour) So far we have considered sequence coalgebras parametrised by an arbitrary set A. In this example we take a special choice, namely $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, the set of decimal digits. We wish to define a coalgebra (or machine) which generates decimal representations of real numbers in the unit interval $[0,1) \subseteq \mathbb{R}$. Notice that this may give rise to both finite sequences ($\frac{1}{8}$ should yield the sequence $\langle 1,2,5\rangle$, for 0.125) and infinite ones ($\frac{1}{3}$ should give $\langle 3,3,3,\ldots\rangle$ for $0.333\ldots$).

The coalgebra we are looking for computes the first decimal of a real number $r \in [0,1)$. Hence it should be of the form

$$[0,1) \xrightarrow{\mathsf{nextdec}} \{\perp\} \cup \Big(\{0,1,2,3,4,5,6,7,8,9\} \times [0,1)\Big)$$

with state space $[0,1)$. How to define $\mathsf{nextdec}$? Especially, when does it stop (i.e. return $\perp$), so that a finite sequence is generated? Well, the decimal representation 0.125 may be identified with $0.12500000\ldots$ with a tail of infinitely many zeros. Clearly, we wish to map such infinitely many zeros to $\perp$. Fair enough, but it does have as consequence that the real number $0 \in [0,1)$ gets represented as the empty sequence.

A little thought brings us to the following:

$$\mathsf{nextdec}(r) = \begin{cases} \perp & \text{if } r = 0 \\ (d, 10r - d) & \text{otherwise, with } d \leq 10r < d+1 \text{ for } d \in A. \end{cases}$$

Notice that this function is well defined, because in the second case the successor state $10r - d$ is within the interval $[0,1)$.

According to the previous proposition, this $\mathsf{nextdec}$ coalgebra gives rise to a behaviour function:

$$[0,1) \xrightarrow{\mathrm{beh}_{\mathsf{nextdec}}} \Big(\{0,1,2,3,4,5,6,7,8,9\}\Big)^{\infty}.$$

In order to understand what it does, i.e. which sequences are generated by $\mathsf{nextdec}$, we consider two examples.

Starting from $\frac{1}{8} \in [0,1)$ we get

$$\begin{aligned} \mathsf{nextdec}(\tfrac{1}{8}) &= (1, \tfrac{1}{4}) \text{ because } 1 \leq \tfrac{10}{8} < 2 \text{ and } \tfrac{10}{8} - 1 = \tfrac{1}{4} \\ \mathsf{nextdec}(\tfrac{1}{4}) &= (2, \tfrac{1}{2}) \text{ because } 2 \leq \tfrac{10}{4} < 3 \text{ and } \tfrac{10}{4} - 2 = \tfrac{1}{2} \\ \mathsf{nextdec}(\tfrac{1}{2}) &= (5, 0) \text{ because } 5 \leq \tfrac{10}{2} < 6 \text{ and } \tfrac{10}{2} - 5 = 0 \\ \mathsf{nextdec}(0) &= \perp. \end{aligned}$$

Thus the resulting $\mathsf{nextdec}$-behaviour on $\frac{1}{8}$ is $\langle 1,2,5\rangle$, i.e. $\mathrm{beh}_{\mathsf{nextdec}}(\frac{1}{8}) = \langle 1,2,5\rangle$. Indeed, in decimal notation we write $\frac{1}{8} = 0.125$.

Next, when we run $\mathsf{nextdec}$ on $\frac{1}{9} \in [0,1)$ we see that

$$\mathsf{nextdec}(\tfrac{1}{9}) = (1, \tfrac{1}{9}) \text{ because } 1 \leq \tfrac{10}{9} < 2 \text{ and } \tfrac{10}{9} - 1 = \tfrac{1}{9}.$$

The function $\mathsf{nextdec}$ thus immediately loops on $\frac{1}{9}$, giving an infinite sequence $\langle 1,1,1,\ldots\rangle$ as behaviour. This corresponds to the fact that we can identify $\frac{1}{9}$ with the infinite decimal representation $0.11111\ldots$.

One sees in the proof of Proposition 1.2.1 that manipulating sequences via their elements is cumbersome and requires us to distinguish between finite and infinite sequences. However, the nice thing about the finality property of A^∞ is that we do not have to work this way anymore. This property states two important aspects, namely *existence* and *uniqueness* of a homomorphism $S \to A^\infty$ into the set of sequences, provided we have a coalgebra structure on S. These two aspects give us two principles:

- **A coinductive definition principle.** The existence aspect tells us how to obtain functions $S \to A^\infty$ into A^∞. If 'recursion' is the appropriate term for definition by induction, then the existence property at hand may be called 'corecursion'.
- **A coinductive proof principle.** The uniqueness aspect tells us how to prove that two functions $f, g\colon S \to A^\infty$ are equal, namely by showing that they are both homomorphisms from a single coalgebra $c\colon S \to \{\perp\} \cup (A \times S)$ to the final coalgebra $\mathsf{next}\colon A^\infty \to \{\perp\} \cup (A \times A^\infty)$.

Coinduction is thus the use of finality – just as induction is the use of initiality, as will be illustrated in Section 2.4. We shall see several examples of the use of these definition and proof principles for sequences in the remainder of this section.

Notation. The previous proposition shows us that coalgebras $c\colon S \to \{\perp\} \cup (A \times S)$ can be understood as generators of sequences, namely via the resulting behaviour function $\mathrm{beh}_c\colon S \to A^\infty$. Alternatively, these coalgebras can be understood as certain automata. The behaviour of a state $x \in S$ of this automaton is then the resulting sequence $\mathrm{beh}_c(x) \in A^\infty$. These sequences $\mathrm{beh}_c(x)$ show only the external behaviour and need not tell everything about states.

Given this behaviour-generating perspective on coalgebras, it will be convenient to use a transition-style notation. For a state $x \in S$ of an arbitrary coalgebra $c\colon S \to \{\perp\} \cup (A \times S)$ we shall often write

$$x \nrightarrow \quad \text{if } c(x) = \perp \qquad \text{and} \qquad x \xrightarrow{a} x' \quad \text{if } c(x) = (a, x'). \tag{1.5}$$

In the first case there is no transition starting from the state x: the automaton c halts immediately at x. In the second case one can do a c-computation starting with x; it produces an observable element $a \in A$ and results in a successor state x'.

This transition notation can be used in particular for the final coalgebra $\mathsf{next}\colon A^\infty \to \{\perp\} \cup (A \times A^\infty)$. In that case, for $\sigma \in A^\infty$, $\sigma \nrightarrow$ means that the sequence σ is empty. In the second case $\sigma \xrightarrow{a} \sigma'$ expresses that the sequence σ can do an a-step to σ', and hence that $\sigma = a \cdot \sigma'$.

Given this new notation we can reformulate the two homomorphism requirements from Proposition 1.2.1 as two implications:

- $x \not\rightarrow \implies \mathrm{beh}_c(x) \not\rightarrow$
- $x \xrightarrow{a} x' \implies \mathrm{beh}_c(x) \xrightarrow{a} \mathrm{beh}_c(x')$.

In the tradition of operational semantics, such implications can also be formulated as rules:

$$\frac{x \not\rightarrow}{\mathrm{beh}_c(x) \not\rightarrow} \qquad\qquad \frac{x \xrightarrow{a} x'}{\mathrm{beh}_c(x) \xrightarrow{a} \mathrm{beh}_c(x')}. \tag{1.6}$$

Such rules thus describe implications: (the conjunction of) what is above the line implies what is below.

In the remainder of this section we consider examples of the use of coinductive definition and proof principles for sequences.

Evenly Listed Elements from a Sequence

Our first aim is to take a sequence $\sigma \in A^\infty$ and turn it into a new sequence $\mathsf{evens}(\sigma) \in A^\infty$ consisting only of the elements of σ at even positions. Step by step we will show how such a function $\mathsf{evens}\colon A^\infty \to A^\infty$ can be defined within a coalgebraic framework, using finality.

Our informal description of $\mathsf{evens}(\sigma)$ can be turned into three requirements:

- If $\sigma \not\rightarrow$, then $\mathsf{evens}(\sigma) \not\rightarrow$, i.e. if σ is empty, then $\mathsf{evens}(\sigma)$ should also be empty.
- If $\sigma \xrightarrow{a} \sigma'$ and $\sigma' \not\rightarrow$, then $\mathsf{evens}(\sigma) \xrightarrow{a} \sigma'$. Thus if σ is the singleton sequence $\langle a \rangle$, then also $\mathsf{evens}(\sigma) = \langle a \rangle$. Notice that by the previous point we could equivalently require $\mathsf{evens}(\sigma) \xrightarrow{a} \mathsf{evens}(\sigma')$ in this case.
- If $\sigma \xrightarrow{a} \sigma'$ and $\sigma' \xrightarrow{a'} \sigma''$, then $\mathsf{evens}(\sigma) \xrightarrow{a} \mathsf{evens}(\sigma'')$. This means that if σ has head a and tail σ', which in its turn has head a' and tail σ'', i.e. if $\sigma = a \cdot a' \cdot \sigma''$, then $\mathsf{evens}(\sigma)$ should have head a and tail $\mathsf{evens}(\sigma'')$, i.e. then $\mathsf{evens}(\sigma) = a \cdot \mathsf{evens}(\sigma'')$. Thus, the intermediate head at odd position is skipped. This is repeated 'coinductively': as long as needed.

As in (1.6) above we can write these three requirements as rules:

$$\frac{\sigma \not\rightarrow}{\mathsf{evens}(\sigma) \not\rightarrow} \quad \frac{\sigma \xrightarrow{a} \sigma' \qquad \sigma' \not\rightarrow}{\mathsf{evens}(\sigma) \xrightarrow{a} \mathsf{evens}(\sigma')} \quad \frac{\sigma \xrightarrow{a} \sigma' \qquad \sigma' \xrightarrow{a'} \sigma''}{\mathsf{evens}(\sigma) \xrightarrow{a} \mathsf{evens}(\sigma'')}. \tag{1.7}$$

One could say that these rules give an 'observational description' of the sequence $\mathsf{evens}(\sigma)$: they describe what we can observe about $\mathsf{evens}(\sigma)$ in terms of what we can observe about σ. For example, if $\sigma = \langle a_0, a_1, a_2, a_3, a_4\rangle$ we can compute

$$\begin{aligned}\mathsf{evens}(\sigma) &= a_0 \cdot \mathsf{evens}(\langle a_2, a_3, a_4\rangle)\\ &= a_0 \cdot a_2 \cdot \mathsf{evens}(\langle a_4\rangle)\\ &= a_0 \cdot a_2 \cdot a_4 \cdot \langle\rangle\\ &= \langle a_0, a_2, a_4\rangle.\end{aligned}$$

Now that we have a reasonable understanding of the function $\mathsf{evens}\colon A^\infty \to A^\infty$ we will see how it arises within a coalgebraic setting. In order to define it coinductively, following the finality mechanism of Proposition 1.2.1, we need to have a suitable coalgebra structure e on the domain A^∞ of the function evens, as in a diagram:

$$\begin{tikzcd}
\{\bot\} \cup (A \times A^\infty) \arrow[r, dashed, "\mathrm{id} \cup (\mathrm{id} \times \mathrm{beh}_e)"] & \{\bot\} \cup (A \times A^\infty) \\
A^\infty \arrow[u, "e"] \arrow[r, dashed, "\mathsf{evens} = \mathrm{beh}_e"'] & A^\infty \arrow[u, "\cong"', "\mathsf{next}"]
\end{tikzcd}$$

That is, for $\sigma \in A^\infty$,

- if $e(\sigma) = \bot$, then $\mathsf{evens}(\sigma) \not\rightarrow$
- if $e(\sigma) = (a, \sigma')$, then $\mathsf{evens}(\sigma) \xrightarrow{a} \mathsf{evens}(\sigma')$.

Combining these two points with the above three rules (1.7) we see that the coalgebra e must be

$$e(\sigma) = \begin{cases} \bot & \text{if } \sigma \not\rightarrow \\ (a, \sigma') & \text{if } \sigma \xrightarrow{a} \sigma' \text{ with } \sigma' \not\rightarrow \\ (a, \sigma'') & \text{if } \sigma \xrightarrow{a} \sigma' \text{ and } \sigma' \xrightarrow{a'} \sigma''. \end{cases}$$

This function e thus tells what can be observed immediately, if anything, and what will be used in the recursion (or corecursion, if you like). It contains the same information as the above three rules. In the terminology used earlier: the coalgebra or automaton e generates the behaviour of evens.

Remark 1.2.3 The coalgebra $e\colon A^\infty \to \{\bot\} \cup (A \times A^\infty)$ illustrates the difference between states and observables. Consider an arbitrary sequence $\sigma \in A^\infty$ and write $\sigma_1 = a \cdot a_1 \cdot \sigma$ and $\sigma_2 = a \cdot a_2 \cdot \sigma$, where $a, a_1, a_2 \in A$ with $a_1 \neq a_2$. These $\sigma_1, \sigma_2 \in A^\infty$ are clearly different states of the coalgebra $e\colon A^\infty \to \{\bot\} \cup (A \times A^\infty)$, but they have the same behaviour: $\mathsf{evens}(\sigma_1) = a \cdot \mathsf{evens}(\sigma) =$

$\mathsf{evens}(\sigma_2)$, where $\mathsf{evens} = \mathrm{beh}_e$. Such observational indistinguishability of the states σ_1, σ_2 is called bisimilarity, written as $\sigma_1 \,\underline{\leftrightarrow}\, \sigma_2$, and will be studied systematically in Chapter 3. Bisimilarity depends on the operations that are available. In this illustration we assume that we have only the coalgebra e.

Oddly Listed Elements from a Sequence

Next we would like to have a similar function $\mathsf{odds}\colon A^\infty \to A^\infty$ which extracts the elements at odd positions. We leave formulation of the appropriate rules to the reader, and claim this function odds can be defined coinductively via the behaviour-generating coalgebra $o\colon A^\infty \to \{\bot\} \cup (A \times A^\infty)$ given by:

$$o(\sigma) = \begin{cases} \bot & \text{if } \sigma \not\rightarrow \text{ or } \sigma \xrightarrow{a} \sigma' \text{ with } \sigma' \not\rightarrow \\ (a', \sigma'') & \text{if } \sigma \xrightarrow{a} \sigma' \text{ and } \sigma' \xrightarrow{a'} \sigma''. \end{cases} \tag{1.8}$$

Thus, we take $\mathsf{odds} = \mathrm{beh}_o$ to be the behaviour function resulting from o, following the finality principle of Proposition 1.2.1. Hence $o(\sigma) = \bot \Rightarrow \mathsf{odds}(\sigma) \not\rightarrow$ and $o(\sigma) = (a, \sigma') \Rightarrow \mathsf{odds}(\sigma) \xrightarrow{a} \mathsf{odds}(\sigma')$. This allows us to compute

$$\begin{aligned} \mathsf{odds}(\langle a_0, a_1, a_2, a_3, a_4\rangle) &= a_1 \cdot \mathsf{odds}(\langle a_2, a_3, a_4\rangle) \\ &\quad\ \text{since } o(\langle a_0, a_1, a_2, a_3, a_4\rangle) = (a_1, \langle a_2, a_3, a_4\rangle) \\ &= a_1 \cdot a_3 \cdot \mathsf{odds}(\langle a_4\rangle) \\ &\quad\ \text{since } o(\langle a_2, a_3, a_4\rangle) = (a_3, \langle a_4\rangle) \\ &= a_1 \cdot a_3 \cdot \langle\rangle \\ &\quad\ \text{since } o(\langle a_4\rangle) = \langle\rangle \\ &= \langle a_1, a_3\rangle. \end{aligned}$$

At this point the reader may wonder: why not define odds via evens, using an appropriate tail function? We shall prove that this gives the same outcome, using coinduction.

Lemma 1.2.4 *One has*

$$\mathsf{odds} = \mathsf{evens} \circ \mathsf{tail},$$

where the function $\mathsf{tail}\colon A^\infty \to A^\infty$ *is given by:*

$$\mathsf{tail}(\sigma) = \begin{cases} \sigma & \text{if } \sigma \not\rightarrow \\ \sigma' & \text{if } \sigma \xrightarrow{a} \sigma'. \end{cases}$$

Proof In order to prove that the two functions $\mathsf{odds}, \mathsf{evens} \circ \mathsf{tail}\colon A^\infty \to A^\infty$ are equal one needs to show by Proposition 1.2.1 that they are both homomorphisms for the same coalgebra structure on A^∞. Since odds arises by definition from the function o in (1.8), it suffices to show that $\mathsf{evens} \circ \mathsf{tail}$ is also a homomorphism from o to next. This involves two points:

- If $o(\sigma) = \perp$, there are two subcases, both yielding the same result:
  - If $\sigma \not\rightarrow$, then $\mathsf{evens}(\mathsf{tail}(\sigma)) = \mathsf{evens}(\sigma) \not\rightarrow$.
  - If $\sigma \xrightarrow{a} \sigma'$ and $\sigma' \not\rightarrow$, then $\mathsf{evens}(\mathsf{tail}(\sigma)) = \mathsf{evens}(\sigma') \not\rightarrow$.
- Otherwise, if $o(\sigma) = (a', \sigma'')$, because $\sigma \xrightarrow{a} \sigma'$ and $\sigma' \xrightarrow{a'} \sigma''$, then we have $\mathsf{evens}(\mathsf{tail}(\sigma)) = \mathsf{evens}(\sigma') \xrightarrow{a'} \mathsf{evens}(\mathsf{tail}(\sigma''))$ since:
  - If $\sigma'' \not\rightarrow$, then $\mathsf{evens}(\sigma') \xrightarrow{a'} \mathsf{evens}(\sigma'') = \mathsf{evens}(\mathsf{tail}(\sigma''))$.
  - If $\sigma'' \xrightarrow{a''} \sigma'''$, then $\mathsf{evens}(\sigma') \xrightarrow{a'} \mathsf{evens}(\sigma''') = \mathsf{evens}(\mathsf{tail}(\sigma''))$. ❑

Such equality proofs using uniqueness may be a bit puzzling at first. But they are very common in category theory and in many other areas of mathematics dealing with universal properties. Later, in Section 3.4 we shall see that such proofs can also be done via bisimulations. This is a common proof technique in process theory – and in coalgebra, of course.

Merging Sequences

In order to further familiarise the reader with the way in which the 'coinductive game' is played, we consider merging two sequences, via a binary operation of the form $\mathsf{merge}\colon A^\infty \times A^\infty \to A^\infty$. We want $\mathsf{merge}(\sigma, \tau)$ to alternatingly take one element from σ and from τ, starting with σ. In terms of rules:

$$\frac{\sigma \not\rightarrow \qquad \tau \not\rightarrow}{\mathsf{merge}(\sigma,\tau) \not\rightarrow} \qquad \frac{\sigma \not\rightarrow \qquad \tau \xrightarrow{a} \tau'}{\mathsf{merge}(\sigma,\tau) \xrightarrow{a} \mathsf{merge}(\sigma,\tau')}$$

$$\frac{\sigma \xrightarrow{a} \sigma'}{\mathsf{merge}(\sigma,\tau) \xrightarrow{a} \mathsf{merge}(\tau,\sigma')}.$$

Notice the crucial reversal of arguments in the last rule.

Thus, the function $\mathsf{merge}\colon A^\infty \times A^\infty \to A^\infty$ is defined coinductively as the behaviour beh_m of the coalgebra

$$(A^\infty \times A^\infty) \xrightarrow{\quad m \quad} \{\perp\} \cup \big(A \times (A^\infty \times A^\infty)\big)$$

given by

$$m(\sigma,\tau) = \begin{cases} \perp & \text{if } \sigma \not\rightarrow \text{ and } \tau \not\rightarrow \\ (a, (\sigma, \tau')) & \text{if } \sigma \not\rightarrow \text{ and } \tau \xrightarrow{a} \tau' \\ (a, (\tau, \sigma')) & \text{if } \sigma \xrightarrow{a} \sigma'. \end{cases}$$

At this stage we can combine all of the coinductively defined functions so far in the following result. It says that the merge of the evenly listed and oddly listed elements in a sequence is equal to the original sequence. At first, this may seem obvious, but recall that our sequences may be finite or infinite, so there is some work to do. The proof is again an exercise in coinductive reasoning using uniqueness. It does not involve a global distinction between finite and infinite but proceeds by local, single step reasoning.

Lemma 1.2.5 *For each sequence $\sigma \in A^\infty$,*

$$\mathit{merge}(\mathit{evens}(\sigma), \mathit{odds}(\sigma)) = \sigma.$$

Proof Let us write $f\colon A^\infty \to A^\infty$ as shorthand for the left-hand side $f(\sigma) = \mathsf{merge}(\mathsf{evens}(\sigma), \mathsf{odds}(\sigma))$. We need to show that f is the identity function. Since the identity function $\mathrm{id}_{A^\infty}\colon A^\infty \to A^\infty$ is a homomorphism from next to next – i.e. $\mathrm{id}_{A^\infty} = \mathrm{beh}_{\mathsf{next}}$ – it suffices to show that also f is such a homomorphism $\mathsf{next} \to \mathsf{next}$. This involves two points:

- If $\sigma \not\to$, then $\mathsf{evens}(\sigma) \not\to$ and $\mathsf{odds}(\sigma) \not\to$; this means that we also have $\mathsf{merge}(\mathsf{evens}(\sigma), \mathsf{odds}(\sigma)) \not\to$ and thus $f(\sigma) \not\to$.
- If $\sigma \xrightarrow{a} \sigma'$, then we distinguish two cases and prove $f(\sigma) \xrightarrow{a} f(\sigma')$ in both, using Lemma 1.2.4.
  - If $\sigma' \not\to$, then $\mathsf{evens}(\sigma) \xrightarrow{a} \mathsf{evens}(\sigma')$ and thus

$$\begin{aligned} f(\sigma) &= \mathsf{merge}(\mathsf{evens}(\sigma), \mathsf{odds}(\sigma)) \\ &\xrightarrow{a} \mathsf{merge}(\mathsf{odds}(\sigma), \mathsf{evens}(\sigma')) \\ &\qquad = \mathsf{merge}(\mathsf{evens}(\mathsf{tail}(\sigma)), \mathsf{evens}(\mathsf{tail}(\sigma'))) \\ &\qquad = \mathsf{merge}(\mathsf{evens}(\sigma'), \mathsf{odds}(\sigma')) \\ &\qquad = f(\sigma'). \end{aligned}$$

  - If $\sigma' \xrightarrow{a'} \sigma''$, then $\mathsf{evens}(\sigma) \xrightarrow{a} \mathsf{evens}(\sigma'')$, and one can derive $f(\sigma) \xrightarrow{a} f(\sigma')$ as before. ❑

We have seen sequences of elements of an arbitrary set A. Things become more interesting when the set A has some algebraic structure, for instance addition or (also) multiplication. Such structure can then be transferred via coinductive definitions to final coalgebras of sequences, leading to what may be called *stream calculus*; see [412].

This completes our introduction to coinduction for sequences. What we have emphasised is that the coalgebraic approach using finality does not consider sequences as a whole via their elements but concentrates on the local, one-step behaviour via head and tail (if any). This makes definitions and reasoning easier – even though the reader may need to see more examples and get more experience to fully appreciate this point. But there is already a clear analogy with induction, which also uses single steps instead of global ones. The formal analogy between induction and coinduction will appear in Section 2.4.

More coinductively defined functions for sequences can be found in [215]. A broader palette of examples is given in [312, 278].

Exercises

1.2.1 Compute the $\mathsf{nextdec}$-behaviour of $\frac{1}{7} \in [0,1)$ as in Example 1.2.2.

1.2.2 Formulate appropriate rules for the function $\mathsf{odds}\colon A^\infty \to A^\infty$ in analogy with the rules (1.7) for evens.

1.2.3 Use coinduction to define the empty sequence $\langle\rangle \in A^\infty$ as a map $\langle\rangle\colon \{\bot\} \to A^\infty$.

Fix an element $a \in A$, and similarly define the infinite sequence $\vec{a}\colon \{\bot\} \to A^\infty$ consisting only of as.

1.2.4 Compute the outcome of $\mathsf{merge}(\langle a_0, a_1, a_2\rangle, \langle b_0, b_1, b_2, b_3\rangle)$.

1.2.5 Is the merge operation associative, i.e. is $\mathsf{merge}(\sigma, \mathsf{merge}(\tau, \rho))$ the same as $\mathsf{merge}(\mathsf{merge}(\sigma, \tau), \rho)$? Give a proof or a counterexample. Is there is neutral element for merge?

1.2.6 Show how to define an alternative merge function which alternatingly takes *two* elements from its argument sequences.

1.2.7
1. Define three functions $\mathsf{ex}_i\colon A^\infty \to A^\infty$, for $i = 0, 1, 2$, which extract the elements at positions $3n + i$.
2. Define $\mathsf{merge3}\colon A^\infty \times A^\infty \times A^\infty \to A^\infty$ satisfying the equation $\mathsf{merge3}(\mathsf{ex}_0(\sigma), \mathsf{ex}_1(\sigma), \mathsf{ex}_2(\sigma)) = \sigma$, for all $\sigma \in A^\infty$.

1.2.8 Consider the sequential composition function $\mathsf{comp}\colon A^\infty \times A^\infty \to A^\infty$ for sequences, described by the three rules:

$$\frac{\sigma \not\to \qquad \tau \not\to}{\mathsf{comp}(\sigma, \tau) \not\to} \qquad \frac{\sigma \not\to \qquad \tau \xrightarrow{a} \tau'}{\mathsf{comp}(\sigma, \tau) \xrightarrow{a} \mathsf{comp}(\sigma, \tau')}$$

$$\frac{\sigma \xrightarrow{a} \sigma'}{\mathsf{comp}(\sigma, \tau) \xrightarrow{a} \mathsf{comp}(\sigma', \tau)}.$$

1. Show by coinduction that the empty sequence $\langle\rangle = \mathsf{next}^{-1}(\bot) \in A^\infty$ is a unit element for comp, i.e. that $\mathsf{comp}(\langle\rangle, \sigma) = \sigma = \mathsf{comp}(\sigma, \langle\rangle)$.
2. Prove also by coinduction that comp is associative, and thus that sequences carry a monoid structure.

1.2.9 Consider two sets A, B with a function $f\colon A \to B$ between them. Use finality to define a function $f^\infty\colon A^\infty \to B^\infty$ that applies f element-wise. Use uniqueness to show that this mapping $f \mapsto f^\infty$ is 'functorial' in the sense that $(\mathrm{id}_A)^\infty = \mathrm{id}_{A^\infty}$ and $(g \circ f)^\infty = g^\infty \circ f^\infty$.

1.2.10 Use finality to define a map $\mathrm{st}\colon A^\infty \times B \to (A \times B)^\infty$ that maps a sequence $\sigma \in A^\infty$ and an element $b \in B$ to a new sequence in $(A \times B)^\infty$ by adding this b at every position in σ. (This is an example of a 'strength' map; see Exercise 2.5.4.)

1.3 Generality of Temporal Logic of Coalgebras

This section will illustrate the important coalgebraic notion of invariant, and use it to introduce temporal operators like $\Box$ for henceforth and $\Diamond$ for eventually. These operators are useful for expressing various interesting properties about states of a coalgebra. As we shall see later in Section 6.4, they can be defined for general coalgebras. But here we shall introduce them in more concrete situations – although we try to suggest the more general perspective. First, the sequences from the previous Section 1.2 will be reconsidered, and next, the statements from Section 1.1 will be used to form a rudimentary notion of class, with associated temporal operators $\Box$ and $\Diamond$ for expressing safety and liveness properties.

1.3.1 Temporal Operators for Sequences

Consider a fixed set A and an arbitrary 'A-sequence' coalgebra $c\colon S \to \{\bot\} \cup (A \times S)$ with state space S. We will be interested in properties of states, expressed via predicates/subsets $P \subseteq S$. For a state $x \in S$ we shall often write $P(x)$ for $x \in P$, and then say that the predicate P holds for x. Such a property $P(x)$ may for instance be: 'the behaviour of x is an infinite sequence'.

For an arbitrary predicate $P \subseteq S$ we shall define several new predicates, namely $\bigcirc P \subseteq S$ for 'nexttime' P, $\Box P \subseteq S$ for 'henceforth' P and $\Diamond P \subseteq S$ for 'eventually' P. These temporal operators $\bigcirc$, $\Box$, $\Diamond$ are all defined with respect to an arbitrary coalgebra $c\colon S \to \{\bot\} \cup (A \times S)$ as above. In order to make this dependence on the coalgebra c explicit we could write $\bigcirc_c P$, $\Box_c P$ and $\Diamond_c P$. But usually it is clear from the context which coalgebra is meant.

All these temporal operators $\bigcirc$, $\Box$, $\Diamond$ talk about future states obtained via transitions to successor states, i.e. via successive applications of the coalgebra. The nexttime operator $\bigcirc$ is most fundamental because it talks about single transitions. The other two, $\Box$ and $\Diamond$, involve multiple steps (zero or more) and are defined in terms of $\bigcirc$. For a sequence coalgebra $c\colon S \to \{\bot\} \cup (A \times S)$ with a predicate $P \subseteq S$ on its state space we define a new predicate $\bigcirc P \subseteq S$, for 'nexttime P', as

$$\begin{aligned}(\bigcirc P)(x) &\iff \forall a \in A.\, \forall x' \in S.\, c(x) = (a, x') \Rightarrow P(x') \\ &\iff \forall a \in A.\, \forall x' \in S.\, x \xrightarrow{a} x' \Rightarrow P(x').\end{aligned} \tag{1.9}$$

In words:

> The predicate $\bigcirc P$ holds for those states x, all of whose successor states x', if any, satisfy P. Thus, $(\bigcirc P)(x)$ indeed means that nexttime after x, P holds.

This simple operator $\bigcirc$ turns out to be fundamental, for example in defining the following notion.

Definition 1.3.1 A predicate P is a (sequence) **invariant** if $P \subseteq \bigcirc P$.

An invariant P is thus a predicate such that if P holds for a state x, then also $\bigcirc P$ holds of x. The latter means that P holds in successor states of x. Hence, if P holds for x, it holds for successors of x. This means that once P holds, P will continue to hold, no matter which transitions are taken. Or, once inside P, one cannot get out.

In general, invariants are important predicates in the study of state-based systems. They often express certain safety or data integrity properties which are implicit in the design of a system; for example, the pressure in a tank will not rise above a certain safety level. An important aspect of formally establishing the safety of systems is to prove that certain crucial predicates are actually invariants.

A concrete example of an invariant on the state space A^∞ of the final sequence coalgebra $\mathsf{next}\colon A^\infty \xrightarrow{\cong} \{\bot\} \cup (A \times A^\infty)$ is the property 'σ is a finite sequence'. Indeed, if σ is finite, and $\sigma \xrightarrow{a} \sigma'$, then also σ' is finite.

Certain predicates $Q \subseteq S$ on the state space of a coalgebra are thus invariants. Given an arbitrary predicate $P \subseteq S$, we can consider those subsets $Q \subseteq P$ which are invariants. The greatest among these subsets plays a special role.

Definition 1.3.2 Let $P \subseteq S$ be an arbitrary predicate on the state space S of a sequence coalgebra.

1. We define a new predicate $\Box P \subseteq S$, for **henceforth** P, to be the greatest invariant contained in P. That is,

$$(\Box P)(x) \iff \exists Q \subseteq S.\, Q \text{ is an invariant } \wedge Q \subseteq P \wedge Q(x).$$

   More concretely, $(\Box P)(x)$ means that all successor states of x satisfy P.

2. And $\Diamond P \subseteq S$, for **eventually** P, is defined as

$$\Diamond P = \neg \Box \neg P,$$

where, for an arbitrary predicate $U \subseteq S$, the negation $\neg U \subseteq S$ is $\{x \in S \mid x \notin U\}$. Hence

$$(\Diamond P)(x) \iff \forall Q \subseteq S. (Q \text{ is an invariant } \wedge Q \subseteq \neg P) \Rightarrow \neg Q(x).$$

Thus, $(\Diamond P)(x)$ says that some successor state of x satisfies P.

The way these temporal operators $\Box$ and $\Diamond$ are defined may seem somewhat complicated at first, but will turn out to be at the right level of abstraction. As we shall see later in Section 6.4, the same formulation in terms of invariants works much more generally for coalgebras of different types (and not just for sequence coalgebras): the definition is 'generic' or 'polytypic'.

In order to show that the abstract formulations in the definition indeed capture the intended meaning of $\Box$ and $\Diamond$ as 'for all future states' and 'for some future state', we prove the following result.

Lemma 1.3.3 *For an arbitrary sequence coalgebra $c: S \to \{\bot\} \cup (A \times S)$, consider its iterations $c^n: S \to \{\bot\} \cup (A \times S)$, for $n \in \mathbb{N}$, as defined in the proof of Proposition 1.2.1. Then, for $P \subseteq S$ and $x \in S$,*

$$\begin{aligned} (\Box P)(x) &\iff P(x) \wedge (\forall n \in \mathbb{N}. \forall a \in A. \forall y \in S. c^n(x) = (a, y) \Rightarrow P(y)) \\ (\Diamond P)(x) &\iff P(x) \vee (\exists n \in \mathbb{N}. \exists a \in A. \exists y \in S. c^n(x) = (a, y) \wedge P(y)). \end{aligned}$$

Proof Since the second equivalence follows by purely logical manipulations from the first one, we shall prove only the first.

($\Rightarrow$) Assume $(\Box P)(x)$, i.e. $Q(x)$ for some invariant $Q \subseteq P$. By induction on $n \in \mathbb{N}$ one gets $c^n(x) = (a, y) \Rightarrow Q(y)$. But then also $P(y)$, for all such y in $c^n(x) = (a, y)$.

($\Leftarrow$) The predicate $\{x \in S \mid P(x) \wedge \forall n \in \mathbb{N}. \forall a \in A. \forall y \in S. c^n(x) = (a, y) \Rightarrow P(y)\}$ is an invariant contained in P. Hence it is contained in $\Box P$. ❑

Example 1.3.4 Consider an arbitrary sequence coalgebra $c: S \to \{\bot\} \cup (A \times S)$. We give three illustrations of the use of temporal operators $\Box$ and $\Diamond$ to express certain properties about states $x \in S$ of this coalgebra c.

1. Recall the termination predicate $(-) \not\to$ introduced in (1.5): $x \not\to$ means $c(x) = \bot$. Now consider the predicate $\Diamond((-) \not\to) \subseteq S$. It holds for those states which are eventually mapped to $\bot$, i.e. for those states whose behaviour is a finite sequence in $A^\star \subseteq A^\infty$.

2. In a similar way we can express that an element $a \in A$ occurs in the behaviour of a state $x \in S$. This is done as

$$\begin{aligned} \mathsf{Occ}(a) &= \Diamond(\{y \in S \mid \exists y' \in S.\, c(y) = (a, y')\}) \\ &= \Diamond(\{y \in S \mid \exists y' \in S.\, y \xrightarrow{a} y'\}). \end{aligned}$$

   One may wish to write $a \in x$ as a more intuitive notation for $x \in \mathsf{Occ}(a)$. It means that there is a future state of x which can do an a-step, i.e. that a occurs somewhere in the behaviour sequence of the state x.
3. Now assume our set A carries an order $\leq$. Consider the predicate:

$$\begin{aligned} &\mathsf{LocOrd}(x) \\ &\iff \forall a, a' \in A.\, \forall x', x'' \in S.\, c(x) = (a, x') \wedge c(x') = (a', x'') \Rightarrow a \leq a' \\ &\iff \forall a, a' \in A.\, \forall x', x'' \in S.\, x \xrightarrow{a} x' \wedge x' \xrightarrow{a'} x'' \Rightarrow a \leq a'. \end{aligned}$$

   Thus, LocOrd holds for x if the first two elements of the behaviour of x, if any, are related by $\leq$. Then,

$$\mathsf{GlobOrd} = \Box\,\mathsf{LocOrd}$$

   holds for those states whose behaviour is an ordered sequence: the elements appear in increasing order.

Next we wish to illustrate how to reason with these temporal operators. We show that an element occurs in the merge of two sequences if and only if it occurs in at least one of the two sequences. Intuitively this is clear, but technically it is not entirely trivial. The proof makes essential use of invariants.

Lemma 1.3.5 *Consider for an element $a \in A$ the occurrence predicate $a \in (-) = \mathsf{Occ}(a) \subseteq A^\infty$ from the previous example, for the final coalgebra $\mathsf{next}\colon A^\infty \xrightarrow{\cong} \{\perp\} \cup (A \times A^\infty)$ from Proposition 1.2.1. Then, for sequences $\sigma, \tau \in A^\infty$,*

$$a \in \mathsf{merge}(\sigma, \tau) \iff a \in \sigma \vee a \in \tau,$$

where $\mathsf{merge}\colon A^\infty \times A^\infty \to A^\infty$ is the merge operator introduced in the previous section.

Proof ($\Rightarrow$) Assume $a \in \mathsf{merge}(\sigma, \tau)$ but neither $a \in \sigma$ nor $a \in \tau$. The latter yields two invariants $P, Q \subseteq A^\infty$ with $P(\sigma)$, $Q(\tau)$ and $P, Q \subseteq \neg\{\rho \mid \exists \rho'.\, \rho \xrightarrow{a} \rho'\}$. These inclusions mean that sequences in P or Q cannot do an a-step.

In order to derive a contradiction we form a new predicate:

$$R = \{\mathsf{merge}(\alpha, \beta) \mid \alpha, \beta \in P \cup Q\}.$$

Clearly, $R(\mathsf{merge}(\sigma, \tau))$. The only transitions that a sequence $\mathsf{merge}(\alpha, \beta) \in R$ can do are

1. $\mathsf{merge}(\alpha,\beta) \xrightarrow{b} \mathsf{merge}(\alpha,\beta')$ because $\alpha \not\to$ and $\beta \xrightarrow{b} \beta'$.
2. $\mathsf{merge}(\alpha,\beta) \xrightarrow{b} \mathsf{merge}(\beta,\alpha')$ because $\alpha \xrightarrow{b} \alpha'$.

In both cases the successor state is again in R, so that R is an invariant. Also, sequences in R cannot do an a-step. The predicate R thus disproves the assumption $a \in \mathsf{merge}(\sigma,\tau)$.

($\Leftarrow$) Assume, without loss of generality, $a \in \sigma$ but not $a \in \mathsf{merge}(\sigma,\tau)$. Thus there is an invariant $P \subseteq \neg\{\rho \mid \exists \rho'.\, \rho \xrightarrow{a} \rho'\}$ with $P(\mathsf{merge}(\sigma,\tau))$. We now take

$$Q = \{\alpha \mid \exists \beta.\, P(\mathsf{merge}(\alpha,\beta)) \vee P(\mathsf{merge}(\beta,\alpha))\}.$$

Clearly $Q(\sigma)$. In order to show that Q is an invariant, assume an element $\alpha \in Q$ with a transition $\alpha \xrightarrow{b} \alpha'$. There are then several cases.

1. If $P(\mathsf{merge}(\alpha,\beta))$ for some β, then $\mathsf{merge}(\alpha,\beta) \xrightarrow{b} \mathsf{merge}(\beta,\alpha')$, so that $\alpha' \in Q$, because $P(\mathsf{merge}(\beta,\alpha'))$, and also $b \neq a$.
2. If $P(\mathsf{merge}(\beta,\alpha))$ for some β, then there are two further cases:

   a. If $\beta \not\to$, then $\mathsf{merge}(\beta,\alpha) \xrightarrow{b} \mathsf{merge}(\beta,\alpha')$, so that $\alpha' \in Q$, and $b \neq a$.

   b. If $\beta \xrightarrow{c} \beta'$, then $\mathsf{merge}(\beta,\alpha) \xrightarrow{c} \mathsf{merge}(\alpha,\beta') \xrightarrow{b} \mathsf{merge}(\alpha',\beta')$. Thus $P(\mathsf{merge}(\alpha',\beta'))$, so that $\alpha' \in Q$, and also $b \neq a$.

These cases also show that Q is contained in $\neg\{\rho \mid \exists \rho'.\, \rho \xrightarrow{a} \rho'\}$. This contradicts the assumption that $a \in \sigma$. ❑

This concludes our first look at temporal operators for sequences, from a coalgebraic perspective.

1.3.2 Temporal Operators for Classes

A class in an object-oriented programming language encapsulates data with associated operations, called 'methods' in this setting. They can be used to access and manipulate the data. These data values are contained in so-called fields or attributes. Using the representation of methods as statements with exceptions E as in Section 1.1 we can describe the operations of a class as a collection of attributes and methods, acting on a state space S:

$$\begin{array}{rcl} \mathsf{at}_1 : S & \longrightarrow & D_1 \\ & \vdots & \\ \mathsf{at}_n : S & \longrightarrow & D_n \\ \mathsf{meth}_1 : S & \longrightarrow & \{\bot\} \cup S \cup (S \times E) \\ & \vdots & \\ \mathsf{meth}_m : S & \longrightarrow & \{\bot\} \cup S \cup (S \times E). \end{array} \tag{1.10}$$

These attributes at_i give the data value $\mathsf{at}_i(x) \in D_i$ in each state $x \in S$. Similarly, each method meth_j can produce a successor state, either normally or exceptionally, in which the attributes have possibly different values. Objects, in the sense of object-oriented programming (not of category theory), are thus identified with states.

For such classes, as for sequences, coalgebraic temporal logic provides a tailor-made nexttime operator $\bigcirc$. For a predicate $P \subseteq S$, we have $\bigcirc P \subseteq S$, defined on $x \in S$ as

$$\begin{aligned}(\bigcirc P)(x) \iff \forall j \leq m.\ &(\forall y \in S.\, \mathsf{meth}_j(x) = y \Rightarrow P(y)) \wedge \\ &(\forall y \in S.\, \forall e \in E.\, \mathsf{meth}_j(x) = (y, e) \Rightarrow P(y)).\end{aligned}$$

Thus, $(\bigcirc P)(x)$ means that P holds in each possible successor state of x, resulting from normal or abrupt termination.

From this point on we can follow the pattern used above for sequences. A predicate $P \subseteq S$ is a **class invariant** if $P \subseteq \bigcirc P$. Also: $\Box P$ is the greatest invariant contained in P, and $\Diamond P = \neg \Box \neg P$. Predicates of the form $\Box P$ are so-called **safety properties** expressing that 'nothing bad will happen': P holds in all future states. And predicates $\Diamond P$ are **liveness properties** saying that 'something good will happen': P holds in some future state.

A typical example of a safety property is: this integer field `i` will always be non-zero (so that it is safe to divide by `i`), or: this array `a` will always be a non-null reference and have length greater than 1 (so that we can safely access `a[0]` and `a[1]`).

Such temporal properties are extremely useful for reasoning about classes. As we have tried to indicate, they arise quite naturally and uniformly in a coalgebraic setting.

Exercises

1.3.1 The nexttime operator $\bigcirc$ introduced in (1.9) is the so-called **weak** nexttime. There is an associated **strong** nexttime, given by $\neg \bigcirc \neg$. Note the difference between weak and strong nexttime for sequences.

1.3.2 Prove that the 'truth' predicate that always holds is a (sequence) invariant. And if P_1 and P_2 are invariants, then so is the intersection $P_1 \cap P_2$. Finally, if P is an invariant, then so is $\bigcirc P$.

1.3.3
1. Show that $\Box$ is an interior operator, i.e. satisfies: $\Box P \subseteq P$, $\Box P \subseteq \Box \Box P$, and $P \subseteq Q \Rightarrow \Box P \subseteq \Box Q$.
2. Prove that a predicate P is an invariant if and only if $P = \Box P$.

1.3.4 Recall the finite behaviour predicate $\Diamond(- \not\rightarrow)$ from Example 1.3.4.1 and show that it is an invariant: $\Diamond(- \not\rightarrow) \subseteq \bigcirc\Diamond(- \not\rightarrow)$. *Hint*: For an invariant Q, consider the predicate $Q' = (\neg(-) \not\rightarrow) \cap (\bigcirc Q)$.

1.3.5 Let $(A, \leq)$ be a complete lattice, i.e. a poset in which each subset $U \subseteq A$ has a join $\bigvee U \in A$. It is well known that each subset $U \subseteq A$ then also has a meet $\bigwedge U \in A$, given by $\bigwedge U = \bigvee\{a \in A \mid \forall b \in U.\, a \leq b\}$.

Let $f : A \to A$ be a monotone function: $a \leq b$ implies $f(a) \leq f(b)$. Recall, e.g. from [119, chapter 4] that such a monotone f has both a least fixed point $\mu f \in A$ and a greatest fixed point $\nu f \in A$ given by the formulas:

$$\mu f = \bigwedge\{a \in A \mid f(a) \leq a\}, \qquad \nu f = \bigvee\{a \in A \mid a \leq f(a)\}.$$

Now let $c : S \to \{\bot\} \cup (A \times S)$ be an arbitrary sequence coalgebra, with associated nexttime operator $\bigcirc$.

1. Prove that $\bigcirc$ is a monotone function $\mathcal{P}(S) \to \mathcal{P}(S)$, i.e. that $P \subseteq Q$ implies $\bigcirc P \subseteq \bigcirc Q$, for all $P, Q \subseteq S$.
2. Check that $\Box P \in \mathcal{P}(S)$ is the greatest fixed point of the function $\mathcal{P}(S) \to \mathcal{P}(S)$ given by $U \mapsto P \cap \bigcirc U$.
3. Define for $P, Q \subseteq S$ a new predicate $P\,\mathcal{U}\,Q \subseteq S$, for '$P$ until Q' as the least fixed point of $U \mapsto Q \cup (P \cap \neg\bigcirc\neg U)$. Check that 'until' is indeed a good name for $P\,\mathcal{U}\,Q$, since it can be described explicitly as

$$\begin{aligned} P\,\mathcal{U}\,Q = \{x \in S \mid \exists n \in \mathbb{N}.\, \exists x_0, x_1, \ldots, x_n \in S.\, & \\ x_0 = x \wedge (\forall i < n.\, \exists a.\, x_i \xrightarrow{a} x_{i+1}) \wedge Q(x_n) & \\ \wedge\, \forall i < n.\, P(x_i)\}. & \end{aligned}$$

   Hint: Don't use the fixed point definition μ, but first show that this subset is a fixed point, and then that it is contained in an arbitrary fixed point.

(The fixed point definitions that we described above are standard in temporal logic; see e.g. [128, 3.24–3.25]. The above operation $\mathcal{U}$ is what is called the 'strong' until. The 'weak' one does not have the negations $\neg$ in its fixed-point description in point 3.)

1.4 Abstractness of the Coalgebraic Notions

In this final section of the chapter we wish to consider the different settings in which coalgebras can be studied. Proper appreciation of the level of generality of coalgebras requires a certain familiarity with the theory of categories. Category theory is a special area that studies the fundamental structures used within mathematics. It is based on the very simple notion of an arrow between objects. Category theory is sometimes described as abstract nonsense, but it is often useful because it provides an abstract framework in which similarities between seemingly different notions become apparent. It has become a standard tool in theoretical computer science, especially in the semantics of programming languages. In particular, the categorical description of fixed points, both of recursive functions and of recursive types, captures the relevant 'universal' properties that are used in programming and reasoning with these constructs. This categorical approach to fixed points forms one of the starting points for the use of category theory in the study of algebras and coalgebras.

For this reason we need to introduce the fundamental notions of category and functor, simply because a bit of category theory helps enormously in presenting the theory of coalgebras and in recognising the common structure underlying many examples. Readers who wish to learn more about categories may consider introductory texts such as [46, 37, 113, 461, 379, 57, 330, 332], or more advanced ones such as [344, 85, 346, 239, 443].

In the beginning of this chapter we have described a coalgebra in (1.1) as a function of the form $\alpha \colon S \to \boxed{\cdots\ S\ \cdots}$ with a structured output type in which the state space S may occur. Here we shall describe such a result type as an expression $F(S) = \boxed{\cdots\ S\ \cdots}$ involving S. Shortly we shall see that F is a functor. A coalgebra is then a map of the form $\alpha \colon S \to F(S)$. It can thus be described in an arrow-theoretic setting, as given by a category.

Definition 1.4.1 A **category** is a mathematical structure consisting of objects with arrows between them, that can be composed.

More formally, a category $\mathbb{C}$ consists of a collection $\mathsf{Obj}(\mathbb{C})$ of objects and a collection $\mathsf{Arr}(\mathbb{C})$ of arrows (also called maps or morphisms). Usually we write $X \in \mathbb{C}$ for $X \in \mathsf{Obj}(\mathbb{C})$. Each arrow in $\mathbb{C}$, written as $X \xrightarrow{f} Y$ or as $f \colon X \to Y$, has a domain object $X \in \mathbb{C}$ and a codomain object $Y \in \mathbb{C}$. These objects and arrows carry a composition structure.

1. For each pair of maps $f \colon X \to Y$ and $g \colon Y \to Z$ there is a composition map $g \circ f \colon X \to Z$. This composition operation $\circ$ is associative: if $h \colon Z \to W$, then $h \circ (g \circ f) = (h \circ g) \circ f$.
2. For each object $X \in \mathbb{C}$ there is an identity map $\mathrm{id}_X \colon X \to X$, such that id is neutral element for composition $\circ$: for $f \colon X \to Y$ one has $f \circ \mathrm{id}_X = f = \mathrm{id}_Y \circ f$. Often, the subscript X in id_X is omitted when it is clear from the context.

Ordinary sets with functions between them form an obvious example of a category, for which we shall write **Sets**. Although **Sets** is a standard example, it is important to realise that a category may be a very different structure. In particular, an arrow in a category need not be a function.

We give several standard examples, and leave it to the reader to check that the requirements of a category hold for all of them.

Example 1.4.2 1. Consider a monoid M with composition operation $+$ and unit element $0 \in M$. This M can also be described as a category with one object, say $\star$, and with arrows $\star \to \star$ given by elements $m \in M$. The identity arrow is then $0 \in M$, and composition of arrows $m_1 \colon \star \to \star$ and $m_2 \colon \star \to \star$ is $m_1 + m_2 \colon \star \to \star$. The associativity and identity requirements required for a category are precisely the associativity and identity laws of the monoid.

2. Here is another degenerate example: a preorder consists of a set D with a reflexive and transitive order relation $\leq$. It corresponds to a category in which there is at most one arrow between each pair of objects. Indeed, the preorder $(D, \leq)$ can be seen as a category with elements $d \in D$ as objects and with an arrow $d_1 \to d_2$ if and only if $d_1 \leq d_2$.
3. Many examples of categories have certain mathematical structures as objects, and structure-preserving functions between them as morphisms. Examples are:

   a. **Mon**, the category of monoids with monoid homomorphisms (preserving composition and unit).
   b. **Grp**, the category of groups with group homomorphisms (preserving composition and unit, and thereby also inverses).
   c. **PreOrd**, the category of preorders with monotone functions (preserving the order). Similarly, there is a category **PoSets** with posets as objects, and also with monotone functions as morphisms.
   d. **Dcpo**, the category of directed complete partial orders (dcpos) with continuous functions between them (preserving the order and directed joins $\bigvee$).
   e. **Sp**, the category of topological spaces with continuous functions (whose inverse image preserves open subsets).
   f. **Met**, the category of metric spaces with non-expansive functions between them. Consider two objects (M_1, d_1) and (M_2, d_2) in **Met**, where $d_i \colon M_i \times M_i \to [0, \infty)$ is a distance function on the set M_i. A morphism $(M_1, d_1) \to (M_2, d_2)$ in **Met** is defined as a function $f \colon M_1 \to M_2$ between the underlying sets satisfying $d_2(f(x), f(y)) \leq d_1(x, y)$, for all $x, y \in M_1$.

4. An example that we shall frequently use is the category **SetsRel** of sets and relations. Its objects are ordinary sets, and its morphisms $X \to Y$ are relations $R \subseteq X \times Y$. Composition of $R\colon X \to Y$ and $S : Y \to Z$ in **SetsRel** is given by relational composition:

$$S \circ R = \{(x, z) \in X \times Z \mid \exists y \in Y.\, R(x, y) \wedge S(y, z)\}. \tag{1.11}$$

The identity morphism $X \to X$ in **SetsRel** is the equality relation (also called diagonal or identity relation) $\mathrm{Eq}(X) \subseteq X \times X$ given by $\mathrm{Eq}(X) = \{(x, x) \mid x \in X\}$.

A category is thus a very general mathematical structure, with many possible instances. In the language of categories one can discuss standard mathematical notions, such as mono-/epi-/iso-morphism, product, limit, etc. For example, an isomorphism in a category $\mathbb{C}$ is a morphism $f\colon X \to Y$ for which there is a (necessarily unique) morphism $g\colon Y \to X$ in the opposite direction with $f \circ g = \mathrm{id}_Y$ and $g \circ f = \mathrm{id}_X$. If there is such an isomorphism, one often writes $X \cong Y$. Such general categorical definitions then have meaning in every example of a category. For instance, it yields a notion of isomorphism for groups, posets, topological spaces, etc.

In a category a morphism from an object to itself, that is, a morphism of the form $f\colon X \to X$, will be called an **endomorphism** or simply an **endomap**. For instance, a relation $R \subseteq X\times X$ on a single set X – instead of a relation $S \subseteq X\times Y$ on two sets X, Y – will be called an endorelation, since it forms an endomap $X \to X$ in the above category **SetsRel**.

Categorical properties are expressed in terms of morphisms, often drawn as diagrams. Two fundamental aspects are commutation and uniqueness.

- **Commutation**: An equation in category theory usually has the form $f_1 \circ \cdots \circ f_n = g_1 \circ \cdots \circ g_m$, for certain morphisms f_i, g_j. Such an equation can be expressed in a commuting diagram as

$$\begin{tikzcd}
& V_1 \arrow[r, "f_2"] & \cdots \arrow[r, "f_{n-1}"] & V_{n-1} \arrow[dr, "f_n"] & \\
X \arrow[ur, "f_1"] \arrow[dr, "g_1"'] & & & & Y \\
& W_1 \arrow[r, "g_2"'] & \cdots \arrow[r, "g_{m-1}"'] & W_{m-1} \arrow[ur, "g_m"'] &
\end{tikzcd}$$

  Extracting such an equation from a commuting diagram by following two paths is an example of what is called *diagram chasing*.

- **Uniqueness**: A frequently occurring formulation is: for every +++ there is a unique morphism $f\colon X \to Y$ satisfying ***. Such uniqueness is often expressed by writing a dashed arrow, $f\colon X \dashrightarrow Y$, especially in a diagram.

As we have already seen in Section 1.2, uniqueness is a powerful reasoning principle: one can derive an equality $f_1 = f_2$ for two morphisms $f_1, f_2\colon X \to Y$ by showing that they both satisfy ***. Often, this property *** can be established via diagram chasing, i.e. by following paths in a diagram (both for f_1 and for f_2).

Both commutation and uniqueness will be frequently used in the course of this book.

For future use we mention two ways to construct new categories from old.

- Given a category $\mathbb{C}$, one can form what is called the **opposite** category $\mathbb{C}^{\mathrm{op}}$ which has the same objects as $\mathbb{C}$, but the arrows reversed. Thus $f\colon X \to Y$ in $\mathbb{C}^{\mathrm{op}}$ if and only if $f\colon Y \to X$ in $\mathbb{C}$. Composition $g \circ f$ in $\mathbb{C}^{\mathrm{op}}$ is then $f \circ g$ in $\mathbb{C}$.
- Given two categories $\mathbb{C}$ and $\mathbb{D}$, we can form the **product** category $\mathbb{C} \times \mathbb{D}$. Its objects are pairs of objects (X, Y) with $X \in \mathbb{C}$ and $Y \in \mathbb{D}$. A morphism $(X, Y) \to (X', Y')$ in $\mathbb{C} \times \mathbb{D}$ consists of a pair of morphisms $X \to X'$ in $\mathbb{C}$ and $Y \to Y'$ in $\mathbb{D}$. Identities and compositions are obtained componentwise.

The above example categories of monoids, groups, etc. indicate that structure-preserving mappings are important in category theory. There is also a notion of such a mapping between categories, called functor. It preserves the relevant structure.

Definition 1.4.3 Consider two categories $\mathbb{C}$ and $\mathbb{D}$. A **functor** $F\colon \mathbb{C} \to \mathbb{D}$ consists of two mappings $\mathsf{Obj}(\mathbb{C}) \to \mathsf{Obj}(\mathbb{D})$ and $\mathsf{Arr}(\mathbb{C}) \to \mathsf{Arr}(\mathbb{D})$, both written as F, such that:

1. F preserves domains and codomains: if $f\colon X \to Y$ in $\mathbb{C}$, then $F(f)\colon F(X) \to F(Y)$ in $\mathbb{D}$.
2. F preserves identities: $F(\mathrm{id}_X) = \mathrm{id}_{F(X)}$ for each $X \in \mathbb{C}$.
3. F preserves composition: $F(g \circ f) = F(g) \circ F(f)$, for all maps $f\colon X \to Y$ and $g\colon Y \to Z$ in $\mathbb{C}$.

For each category $\mathbb{C}$ there is a trivial 'identity' functor $\mathrm{id}_{\mathbb{C}}\colon \mathbb{C} \to \mathbb{C}$, mapping $X \mapsto X$ and $f \mapsto f$. Also, for each object $A \in \mathbb{C}$ there are functors which map everything to A. They can be defined as functors $A\colon \mathbb{D} \to \mathbb{C}$, for an arbitrary category $\mathbb{D}$. This constant functor maps any object $X \in \mathbb{D}$ to A, and any morphism f in $\mathbb{D}$ to the identity map $\mathrm{id}_A\colon A \to A$.

Further, given two functors $F\colon \mathbb{C} \to \mathbb{D}$ and $G\colon \mathbb{D} \to \mathbb{E}$, there is a composite functor $G \circ F\colon \mathbb{C} \to \mathbb{E}$. It is given by $X \mapsto G(F(X))$ and $f \mapsto G(F(f))$. Often we simply write $GF = G \circ F$, and similarly $GF(X) = G(F(X))$.

Example 1.4.4 1. Consider two monoids $(M, +, 0)$ and $(N, \cdot, 1)$ as categories, as in Example 1.4.2.1. A functor $f\colon M \to N$ is then the same as a monoid homomorphism: it preserves the composition operation and unit element.

2. Similarly, consider two preorders $(D, \leq)$ and $(E, \sqsubseteq)$ as categories, as in Example 1.4.2.3. A functor $f\colon D \to E$ is then nothing but a monotone function: $x \leq y$ implies $f(x) \sqsubseteq f(y)$.
3. Frequently occurring examples of functors are so-called **forgetful** functors. They forget part of the structure of their domain. For instance, there is a forgetful functor **Mon** $\to$ **Sets** mapping a monoid $(M, +, 0)$ to its underlying set M, and mapping a monoid homomorphism f to f, considered as a function between sets. Similarly, there is a forgetful functor **Grp** $\to$ **Mon** mapping groups to monoids by forgetting their inverse operation.
4. There is a 'graph' functor **Sets** $\to$ **SetsRel**. It maps a set X to X itself, and a function $f\colon X \to Y$ to the corresponding graph relation $\mathrm{Graph}(f) \subseteq X \times Y$ given by $\mathrm{Graph}(f) = \{(x, y) \mid f(x) = y\}$.
5. Recall from Section 1.2 that sequence coalgebras were described as functions of the form $c\colon S \to \{\bot\} \cup (A \times S)$. Their codomain can be described via a functor $\mathsf{Seq}\colon$ **Sets** $\to$ **Sets**. It maps a set X to the set $\{\bot\} \cup (A \times X)$. And it sends a function $f\colon X \to Y$ to a function $\{\bot\} \cup (A \times X) \to \{\bot\} \cup (A \times Y)$ given by

$$\bot \longmapsto \bot \qquad \text{and} \qquad (a, x) \longmapsto (a, f(x)).$$

We leave it to the reader to check that Seq preserves compositions and identities. We do note that the requirement that the behaviour function $\mathrm{beh}_c\colon S \to A^\infty$ from Proposition 1.2.1 is a homomorphism of coalgebras can now be described via commutation of the following diagram:

$$\begin{tikzcd}
\mathsf{Seq}(S) \arrow[r, dashed, "\mathsf{Seq}(\mathrm{beh}_c)"] & \mathsf{Seq}(A^\infty) \\
S \arrow[u, "c"] \arrow[r, dashed, "\mathrm{beh}_c"'] & A^\infty \arrow[u, "\cong"', "\mathrm{next}"]
\end{tikzcd}$$

In this book we shall be especially interested in **endofunctors**, i.e. in functors $\mathbb{C} \to \mathbb{C}$ from a category $\mathbb{C}$ to itself. In many cases this category $\mathbb{C}$ will simply be **Sets**, the category of sets and functions. Often we say that a mapping $A \mapsto G(A)$ of sets to sets is **functorial** if it can be extended in a more or less obvious way to a mapping $f \mapsto G(f)$ on functions such that G becomes a functor $G\colon$ **Sets** $\to$ **Sets**. We shall see many examples in the next chapter.

We can now introduce coalgebras in full generality.

Definition 1.4.5 Let $\mathbb{C}$ be an arbitrary category, with an endofunctor $F\colon \mathbb{C} \to \mathbb{C}$.

1. An F-**coalgebra**, or just a **coalgebra** when F is understood, consists of an object $X \in \mathbb{C}$ together with a morphism $c\colon X \to F(X)$. As before, we often call X the state space, or the carrier of the coalgebra, and c the transition or coalgebra structure.
2. A **homomorphism of coalgebras**, or a **map of coalgebras** or a **coalgebra map**, from one coalgebra $c\colon X \to F(X)$ to another coalgebra $d\colon Y \to F(Y)$ consists of a morphism $f\colon X \to Y$ in $\mathbb{C}$ which commutes with the structures, in the sense that the following diagram commutes:

$$\begin{tikzcd}
F(X) \arrow[r, "F(f)"] & F(Y) \\
X \arrow[u, "c"] \arrow[r, "f"'] & Y \arrow[u, "d"']
\end{tikzcd}$$

3. F-coalgebras with homomorphisms between them form a category, which we shall write as $\mathbf{CoAlg}(F)$. It comes with a forgetful functor $\mathbf{CoAlg}(F) \to \mathbb{C}$, mapping a coalgebra $X \to F(X)$ to its state space X, and a coalgebra homomorphism f to f.

The abstractness of the notion of coalgebra lies in the fact that it can be expressed in any category. So we not only need to talk about coalgebras in **Sets**, as we have done so far, but can also consider coalgebras in other categories. For instance, one can have coalgebras in **PreOrd**, the category of preorders. In that case, the state space is a preorder, and the coalgebra structure is a monotone function. Similarly, a coalgebra in the category **Mon** of monoids has a monoid as state space, and a structure which preserves this monoid structure. We can even have a coalgebra in a category $\mathbf{CoAlg}(F)$ of coalgebras. We briefly mention some examples, without going into details.

- Real numbers (and also Baire and Cantor space) are described in [378, theorem 5.1] as final coalgebras (via continued fractions; see also [366]) of an endofunctor on the category **PoSets**.
- So-called descriptive general frames (special models of modal logic) appear in [316] as coalgebras of the Vietoris functor on the category of Stone spaces.

- At several places in this book we shall see coalgebra of endofunctors other than sets. For instance, Exercise 1.4.6 mentions invariants as coalgebras of endofunctors on poset categories, and Example 2.3.10 and Exercise 2.3.7 describe streams with their topology as final coalgebra in the category of topological spaces. Section 5.3 introduces traces of suitable coalgebras via coalgebra homomorphism to a final coalgebra in the category **SetsRel** of sets with relations as morphisms.

In the next few chapters we shall concentrate on coalgebras in **Sets**, but occasionally this more abstract perspective will be useful.

Exercises

1.4.1 Let $(M, +, 0)$ be a monoid, considered as a category. Check that a functor $F\colon M \to \mathbf{Sets}$ can be identified with a **monoid action**: a set X together with a function $\mu\colon X \times M \to X$ with $\mu(x, 0) = x$ and $\mu(x, m_1 + m_2) = \mu(\mu(x, m_2), m_1)$.

1.4.2 Check in detail that the opposite $\mathbb{C}^{\mathrm{op}}$ and product $\mathbb{C} \times \mathbb{D}$ are indeed categories.

1.4.3 Assume an arbitrary category $\mathbb{C}$ with an object $I \in \mathbb{C}$. We form a new category $\mathbb{C}/I$, the so-called **slice category** over I, with

objects	maps $f\colon X \to I$ with codomain I in $\mathbb{C}$
morphisms	from $X \xrightarrow{f} I$ to $Y \xrightarrow{g} I$ are morphisms $h\colon X \to Y$ in $\mathbb{C}$ for which the following diagram commutes:

$$\begin{tikzcd} X \arrow[rr, "h"] \arrow[dr, "f"'] & & Y \arrow[dl, "g"] \\ & I & \end{tikzcd}$$

1. Describe identities and composition in $\mathbb{C}/I$, and verify that $\mathbb{C}/I$ is a category.
2. Check that taking domains yields a functor $\mathrm{dom}\colon \mathbb{C}/I \to \mathbb{C}$.
3. Verify that for $\mathbb{C} = \mathbf{Sets}$, a map $f\colon X \to I$ may be identified with an I-indexed family of sets $(X_i)_{i\in I}$, namely where $X_i = f^{-1}(i)$. What do morphisms in $\mathbb{C}/I$ correspond to, in terms of such indexed families?

1.4.4 Recall that for an arbitrary set A we write $A^\star$ for the set of finite sequences $\langle a_0, \ldots, a_n\rangle$ of elements $a_i \in A$.

1. Check that $A^\star$ carries a monoid structure given by concatenation of sequences, with the empty sequence $\langle\rangle$ as neutral element.
2. Check that the assignment $A \mapsto A^\star$ yields a functor **Sets** $\to$ **Mon** by mapping a function $f : A \to B$ between sets to the function $f^\star : A^\star \to B^\star$ given by $\langle a_0, \ldots, a_n\rangle \mapsto \langle f(a_0), \ldots, f(a_n)\rangle$. (Be aware of what needs to be checked: $f^\star$ must be a monoid homomorphism, and $(-)^\star$ must preserve composition of functions and identity functions.)
3. Prove that $A^\star$ is the **free monoid on** A: there is the singleton-sequence insertion map $\eta : A \to A^\star$ which is universal among all mappings of A into a monoid. The latter means that for each monoid $(M, 0, +)$ and function $f : A \to M$ there is a unique monoid homomorphism $g : A^\star \to M$ with $g \circ \eta = f$.

1.4.5 Recall from (1.3) the statements with exceptions of the form $S \to \{\perp\} \cup S \cup (S \times E)$.

1. Prove that the assignment $X \mapsto \{\perp\} \cup X \cup (X \times E)$ is functorial, so that statements are coalgebras for this functor.
2. Show that all the operations $\mathsf{at}_1, \ldots, \mathsf{at}_n, \mathsf{meth}_1, \ldots, \mathsf{meth}_m$ of a class as in (1.10) can also be described as a single coalgebra, namely of the functor:

$$X \mapsto D_1 \times \cdots \times D_n \times \underbrace{(\{\perp\} \cup X \cup (X \times E)) \times \cdots \times (\{\perp\} \cup X \cup (X \times E))}_{m \text{ times}}\ .$$

1.4.6 Recall the nexttime operator $\bigcirc$ for a sequence coalgebra $c : S \to \mathsf{Seq}(S) = \{\perp\} \cup (A \times S)$ from the previous section. Exercise 1.3.5.1 says that it forms a monotone function $\mathcal{P}(S) \to \mathcal{P}(S)$ – with respect to the inclusion order – and thus a functor. Check that invariants are precisely the $\bigcirc$-coalgebras!