1

An introduction to (co)algebra and (co)induction

1.1 Introduction

Algebra is a well-established part of mathematics, dealing with sets with operations satisfying certain properties, like groups, rings , vector spaces, etcetera. Its results are essential throughout mathematics and other sciences. *Universal* algebra is a part of algebra in which algebraic structures are studied at a high level of abstraction and in which general notions like homomorphism, subalgebra, congruence are studied in themselves, see *e.g.* [Coh81, MT92, Wec92]. A further step up the abstraction ladder is taken when one studies algebra with the notions and tools from category theory. This approach leads to a particularly concise notion of what is an algebra (for a functor or for a monad), see for example [Man74]. The conceptual world that we are about to enter owes much to this categorical view, but it also takes inspiration from universal algebra, see *e.g.* [Rut00].

In general terms, a program in some programming language manipulates data. During the development of computer science over the past few decades it became clear that an abstract description of these data is desirable, for example to ensure that one's program does not depend on the particular representation of the data on which it operates. Also, such abstractness facilitates correctness proofs. This desire led to the use of algebraic methods in computer science, in a branch called *algebraic specification* or *abstract data type theory*. The object of study are data types in themselves, using notions and techniques which are familiar from algebra. The data types used by computer scientists are often generated from a given collection of (constructor) operations. The same applies in fact to programs, which themselves can be viewed as data too. It is for this reason that "initiality" of algebras plays such an important role in computer science (as first clearly emphasised in [GTW78]). See for example [EM85, Wir90, Wec92] for more information.

Standard algebraic techniques have proved useful in capturing various essential aspects of data structures used in computer science. But it turned out to be difficult to algebraically describe some of the inherently dynamical structures occuring in computing. Such structures usually involve a notion of state, which can be transformed in various ways. Formal approaches to such state-based dynamical systems generally make use of automata or transition systems, see *e.g.* [Plo81, Par81, Mil89] as classical early references. During the last decade the insight gradually grew that such state-based systems should not be described as algebras, but as so-called coalgebras. These are the formal duals of algebras, in a way which will be made precise in this tutorial. The dual property of initiality for algebras, namely finality, turned out to be crucial for such coalgebras. And the logical reasoning principle that is needed for such final coalgebras is not induction but coinduction.

These notions of coalgebra and coinduction are still relatively unfamiliar, and it is our aim in this tutorial to explain them in elementary terms. Most of the literature already assumes some form of familiarity either with category theory, or with the (dual) coalgebraic way of thinking (or both).

Before we start, we should emphasise that there is no new (research) material in this tutorial. Everything that we present is either known in the literature, or in the folklore, so we do not have any claims to originality. Also, our main concern is with conveying ideas, and not with giving a correct representation of the historical developments of these ideas. References are given mainly in order to provide sources for more (background) information.

Also, we should emphasise that we do not assume any knowledge of category theory on the part of the reader. We shall often use the diagrammatic notation which is typical of category theory, but only in order to express equality of two composites of functions, as often used also in other contexts. This is simply the most efficient and most informative way of presenting such information. But in order to fully appreciate the underlying duality between algebra and induction on the one hand, and coalgebra and coinduction on the other, some elementary notions from category theory are needed, especially the notions of *functor* (homomorphism of categories), and of *initial* and *final* (also called *terminal*) object in a category. Here we shall explain these notions in the concrete set-theoretic setting in which we are working, but we definitely encourage the interested reader who wishes to further pursue the topic of this tutorial to study category theory in greater detail. Among the many available texts on category theory, [Pie91, Wal91, AM75, Awo06] are recommended as easy-going starting points, [BW90, Cro93, LS86] as more substantial texts, and [Lan71, Bor94] as advanced reference texts.

This tutorial starts with some introductory expositions in Sections 1.2 – 1.4. The technical material in the subsequent sections is organised as follows.

(1) The starting point is ordinary induction, both as a definition principle and as a proof principle. We shall assume that the reader is familiar with induction, over natural numbers, but also over other data types, say of lists, trees or (in general) of terms. The first real step is to reformulate ordinary induction in a more abstract way, using initiality (see Section 1.5). More precisely, using initiality for "algebras of a functor". This is something which we do not assume to be familiar. We therefore explain how signatures of operations give rise to certain functors, and how algebras of these functors correspond to algebras (or models) of the signatures (consisting of a set equipped with certain functions interpreting the operations). This description of induction in terms of algebras (of functors) has the advantage that it is highly generic, in the sense that it applies in the same way to all kinds of (algebraic) data types. Further, it can be dualised easily, thus giving rise to the theory of coalgebras.

(2) The dual notion of an algebra (of a functor) is a coalgebra (of a functor). It can also be understood as a model consisting of a set with certain operations, but the direction of these operations is not as in algebra. The dual notion of initiality is finality, and this finality gives us coinduction, both as a definition principle and as a reasoning principle. This pattern is as in the previous point, and is explained in Section 1.6.

(3) In Section 1.7 we give an alternative formulation of the coinductive reasoning principle (introduced in terms of finality) which makes use of bisimulations. These are relations on coalgebras which are suitably closed under the (coalgebraic) operations; they may be understood as duals of congruences, which are relations which are closed under algebraic operations. Bisimulation arguments are used to prove the equality of two elements of a final coalgebra, and require that these elements are in a bisimulation relation.

(4) In Section 1.8 we present a coalgebraic account of transition systems and a simple calculus of *processes*. The latter will be defined as the elements of a final coalgebra. An elementary language for the construction of processes will be introduced and its semantics will be defined coinductively. As we shall see, this will involve the mixed occurrence of both algebraic and coalgebraic structures. The combination of algebra and coalgebra will also play a central role in Section 1.9, where a coalgebraic description is given of trace semantics.

In a first approximation, the duality between induction and coinduction that we intend to describe can be understood as the duality between least and greatest fixed points (of a monotone function), see Exercise 1.10.3. These notions generalise to least and greatest fixed points of a functor, which are suitably described as initial algebras and final coalgebras. The point of view mentioned in (1) and (2) above can be made more explicit as follows—without going into technicalities yet. The abstract reformulation of induction that we will describe is:

$$\boxed{\text{induction} \quad = \quad \text{use of initiality for algebras}}$$

An algebra (of a certain kind) is *initial* if for an arbitrary algebra (of the same kind) there is a unique homomorphism (structure-preserving mapping) of algebras:

$$\begin{pmatrix} \text{initial} \\ \text{algebra} \end{pmatrix} \xrightarrow[\text{homomorphism}]{\text{unique}} \begin{pmatrix} \text{arbitrary} \\ \text{algebra} \end{pmatrix} \tag{1.1}$$

This principle is extremely useful. Once we know that a certain algebra is initial, by this principle we can define functions acting on this algebra. Initiality involves unique existence, which has two aspects:

Existence. This corresponds to (ordinary) *definition* by induction.

Uniqueness. This corresponds to *proof* by induction. In such uniqueness proofs, one shows that two functions acting on an initial algebra are the same by showing that they are both homomorphisms (to the same algebra).

The details of this abstract reformulation will be elaborated as we proceed.

Dually, coinduction may be described as:

$$\boxed{\text{coinduction} \quad = \quad \text{use of finality for coalgebras}}$$

A coalgebra (of some kind) is *final* if for an arbitrary coalgebra (of the same kind), there is a unique homomorphism of coalgebras as shown:

$$\begin{pmatrix} \text{arbitrary} \\ \text{coalgebra} \end{pmatrix} \xrightarrow[\text{homomorphism}]{\text{unique}} \begin{pmatrix} \text{final} \\ \text{coalgebra} \end{pmatrix} \tag{1.2}$$

Again we have the same two aspects: existence and uniqueness, corresponding this time to definition and proof by coinduction.

The initial algebras and final coalgebras which play such a prominent role in this theory can be described in a canonical way: an initial algebra can be obtained from the closed terms (*i.e.* from those terms which are generated by iteratively applying the algebra's constructor operations), and the final coalgebra can be obtained from the pure observations. The latter is probably not very familiar, and will be illustrated in several examples in Section 1.2.

History of this chapter: An earlier version of this chapter was published as "*A Tutorial on (Co)(Algebras and (Co)Induction*", in: EATCS Bulletin 62 (1997), p.222–259. More then ten years later, the present version has been updated. Notably, two sections have been added that are particularly relevant for the context of the present book: Processes coalgebraically (Section 1.8), and: Trace semantics coalgebraically (Section 1.9). In both these sections both initial algebras and final coalgebras arise in a natural combination. In addition, the references to related work have been brought up-to-date.

Coalgebra has by now become a well-established part of the foundations of computer science and (modal) logic. In the last decade, much new coalgebraic theory has been developed, such as so-called universal coalgebra [Rut00, Gum99], in analogy to universal algebra, and coalgebraic logic, generalising in various ways classical modal logic, see for instance [Kur01, Kur06, CP07, Kli07] for an overview. But there is much more, none of which is addressed in any detail here. Much relevant recent work and many references can be found in the proceedings of the workshop series CMCS: *Coalgebraic Methods in Computer Science* (published in the ENTCS series) and CALCO: *Conference on Algebra and Coalgebra in Computer Science* (published in the LNCS series). The aim of this tutorial is in essence still the same as it was ten years ago: to provide a brief introduction to the field of coalgebra.

1.2 Algebraic and coalgebraic phenomena

The distinction between algebra and coalgebra pervades computer science and has been recognised by many people in many situations, usually in terms of data versus machines. A modern, mathematically precise way to express the difference is in terms of algebras and coalgebras. The basic dichotomy may be described as *construction* versus *observation*. It may be found in process theory [Mil89], data type theory [GGM76, GM82, AM82, Kam83] (including the theory of classes and objects in object-oriented programming [Rei95, HP95, Jac96b, Jac96a]), semantics of programming languages [MA86] (denotational versus operational [RT94, Tur96, BV96]) and of lambda-calculi [Pit94, Pit96, Fio96, HL95], automata theory [Par81], system theory [Rut00], natural language theory [BM96, Rou96] and many other fields.

We assume that the reader is familiar with definitions and proofs by (ordinary) induction. As a typical example, consider for a fixed data set A, the set $A^\star = \mathsf{list}(A)$ of finite sequences (lists) of elements of A. One can inductively define a length function $\mathsf{len}\colon A^\star \to \mathbb{N}$ by the two clauses:

$$\mathsf{len}(\langle\rangle) = 0 \qquad \text{and} \qquad \mathsf{len}(a \cdot \sigma) = 1 + \mathsf{len}(\sigma)$$

for all $a \in A$ and $\sigma \in A^\star$. Here we have used the notation $\langle\rangle \in A^\star$ for the empty list (sometimes called nil), and $a \cdot \sigma$ (sometimes written as $\mathsf{cons}(a, \sigma)$) for the list obtained from $\sigma \in A^\star$ by prefixing $a \in A$. As we shall see later, the definition of this length function $\mathsf{len}\colon A^\star \to \mathbb{N}$ can be seen as an instance of the above initiality diagram (1.1).

A typical induction proof that a predicate $P \subseteq A^\star$ holds for all lists requires us to prove the induction assumptions

$$P(\langle\rangle) \qquad \text{and} \qquad P(\sigma) \Rightarrow P(a \cdot \sigma)$$

for all $a \in A$ and $\sigma \in A^\star$. For example, in this way one can prove that $\mathsf{len}(\sigma \cdot a) = 1 + \mathsf{len}(\sigma)$ by taking $P = \{\sigma \in A^\star \mid \forall a \in A.\ \mathsf{len}(\sigma \cdot a) = 1 + \mathsf{len}(\sigma)\}$. Essentially, this induction proof method says that $A^\star$ has no proper subalgebras. In this (algebraic) setting we make use of the fact that all finite lists of elements of A can be constructed from the two operations $\mathsf{nil} \in A^\star$ and $\mathsf{cons}\colon A \times A^\star \to A^\star$. As above, we also write $\langle\rangle$ for nil and $a \cdot \sigma$ for $\mathsf{cons}(a, \sigma)$.

Next we describe some typically coalgebraic phenomena, by sketching some relevant examples. Many of the issues that come up during the description of these examples will be explained in further detail in later sections.

(i) Consider a black-box machine (or process) with one (external) button and one light. The machine performs a certain action only if the button is pressed. And the light goes on only if the machine stops operating (*i.e.* has reached a final state); in that case, pressing the button has no effect any more. A client on the outside of such a machine cannot directly observe the internal state of the machine, but (s)he can only observe its behaviour via the button and the light. In this simple (but paradigmatic) situation, all that can be observed directly about a particular state of the machine is whether the light is on or not. But a user may iterate this experiment, and record the observations after a change of state caused by pressing the button[1]. In this situation, a user can observe how many times (s)he has to press the button to make the light go on. This may be zero times (if the light is already on), $n \in \mathbb{N}$ times, or infinitely many times (if the machine keeps on operating and the light never goes on).

Mathematically, we can describe such a machine in terms of a set X, which we understand as the unknown state space of the machine, on which we have a function

$$\mathsf{button}\colon X \longrightarrow \{*\} \cup X$$

where $*$ is a new symbol not occurring in X. In a particular state $s \in X$, apply-

[1] It is assumed that such actions of pressing a button happen instantaneously, so that there is always an order in the occurrence of such actions.

ing the function button—which corresponds to pressing the button—has two possible outcomes: either $\mathsf{button}(s) = *$, meaning that the machine stops operating and that the light goes on, or $\mathsf{button}(s) \in X$. In the latter case the machine has moved to a next state as a result of the button being pressed. (And in this next state, the button can be pressed again) The above pair $(X, \mathsf{button}\colon X \to \{*\} \cup X)$ is an example of a coalgebra.

The observable behaviour resulting from iterated observations as described above yields an element of the set $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$, describing the number of times the button has to be pressed to make the light go on. Actually, we can describe this behaviour as a function $\mathsf{beh}\colon X \to \overline{\mathbb{N}}$. As we shall see later, it can be obtained as instance of the finality diagram (1.2).

(ii) Let us consider a slightly different machine with two buttons: value and next. Pressing the value button results in some visible indication (or attribute) of the internal state (*e.g.* on a display), taking values in a dataset A, without affecting the internal state. Hence pressing value twice consecutively yields the same result. By pressing the next button the machine moves to another state (the value of which can be inspected again). Abstractly, this new machine can be described as a coalgebra

$$\langle \mathsf{value}, \mathsf{next} \rangle\colon X \longrightarrow A \times X$$

on a state space X. The behaviour that we can observe of such machines is the following: for all $n \in \mathbb{N}$, read the value after pressing the next button n times. This results in an infinite sequence $(a_0, a_1, a_2, \ldots) \in A^{\mathbb{N}}$ of elements of the dataset A, with element a_i describing the value after pressing next i times. Observing this behaviour for every state $s \in X$ gives us a function $\mathsf{beh}\colon X \to A^{\mathbb{N}}$.

The set $A^{\mathbb{N}}$ of infinite sequences, in computer science also known as streams, carries itself a coalgebra structure

$$\langle \mathsf{head}, \mathsf{tail} \rangle : A^{\mathbb{N}} \to A \times A^{\mathbb{N}}$$

given, for all $\alpha = (a_0, a_1, a_2, \ldots) \in A^{\mathbb{N}}$, by

$$\mathsf{head}(\alpha) = a_0 \qquad \mathsf{tail}(\alpha) = (a_1, a_2, a_3, \ldots)$$

This coalgebra is final and the behaviour function $\mathsf{beh}\colon X \to A^{\mathbb{N}}$ can thus be seen as an instance of (1.2).

(iii) The previous example is leading us in the direction of a coalgebraic description of classes in object-oriented languages. Suppose we wish to capture the essential aspects of the class of points in a (real) plane that can be moved around by a client. In this situation we certainly want two attribute buttons $\mathsf{first}\colon X \to \mathbb{R}$ and $\mathsf{second}\colon X \to \mathbb{R}$ which tell us, when pushed, the first and second coordinate of a point belonging to this class. As before, the X plays the role of a hidden state space, and elements of X are seen as objects of the class (so that an object is identified with a state). Further we want a button (or method, in object-oriented terminology) $\mathsf{move}\colon X \times (\mathbb{R} \times \mathbb{R}) \to X$ which requires two parameters (corresponding to the change in first and second coordinate). This move operation allows us to change a state in a certain way, depending on the values of the parameters. The move method can equivalently be described as a function $\mathsf{move}\colon X \to X^{(\mathbb{R}\times\mathbb{R})}$ taking the state as single argument, and yielding a function $(\mathbb{R} \times \mathbb{R}) \to X$ from parameters to states.

As a client of such a class we are not interested in the actual details of the implementation (what the state space X exactly looks like) as long as the behaviour is determined by the following two equations:

$$\begin{array}{rcl} \mathsf{first}(\mathsf{move}(s,(d1,d2))) & = & \mathsf{first}(s) + d1 \\ \mathsf{second}(\mathsf{move}(s,(d1,d2))) & = & \mathsf{second}(s) + d2 \end{array}$$

These describe the first and second coordinates after a move in terms of the original coordinates and the parameters of the move. Such equations can be seen as constraints on the observable behaviour.

An important aspect of the object-oriented approach is that classes are built around a hidden state space, which can only be observed and modified via certain specified operations. A user is not interested in the details of the actual implementation, but only in the behaviour that is realised. This is why our black-box description of classes with an unknown state space X is appropriate.

The three buttons of such a class (as abstract machine) can be combined into a single function

$$\langle \mathsf{first}, \mathsf{second}, \mathsf{move} \rangle\colon X \longrightarrow \mathbb{R} \times \mathbb{R} \times X^{(\mathbb{R}\times\mathbb{R})}$$

which forms a coalgebra on the state space X. The observable behaviour is very simple in this case. It consists of the values of the first and second coordinates, since if we know these values, then we know the future observable behaviour: the only change of state that we can bring about is through the move button; but its observable effect is determined by the above two equations. Thus what we can observe about a state is obtained by direct observation, and repeated observations do not produce new information. Hence our behaviour function takes the form $\mathsf{beh}\colon X \to \mathbb{R} \times \mathbb{R}$, and is again an instance of (1.2)[1]. In automata-theoretic terms one can call the space $\mathbb{R} \times \mathbb{R}$ the minimal realisation (or implementation) of the specified behaviour.

In the above series of examples of coalgebras we see each time a state space X

[1] To be precise, for coalgebras of a comonad.

about which we make no assumptions. On this state space a function is defined of the form

$$f : X \to \boxed{\quad X \quad}$$

where the box on the right is some expression involving X again. Later this will be identified as a *functor*. The function f often consists of different components, which allow us either to observe some aspect of the state space directly, or to move on to next states. We have limited access to this state space in the sense that we can only observe or modify it via these specified operations. In such a situation all that we can describe about a particular state is its behaviour, which arises by making successive observations. This will lead to the notion of bisimilarity of states: it expresses of two states that we cannot distinguish them via the operations that are at our disposal, *i.e.* that they are "equal as far as we can see". But this does not mean that these states are also identical as elements of X. Bisimilarity is an important, and typically coalgebraic, concept.

The above examples are meant to suggest the difference between construction in algebra, and observation in coalgebra. This difference will be described more formally below. In practice it is not always completely straightforward to distinguish between algebraic and coalgebraic aspects, for the following two reasons.

(1) Certain abstract operations, like $X \times A \to X$, can be seen as both algebraic and coalgebraic. Algebraically, such an operation allows us to build new elements in X starting from given elements in X and parameters in A. Coalgebraically, this operation is often presented in the equivalent from $X \to X^A$ using function types. It is then seen as acting on the state space X, and yielding for each state a function from A to X which produces for each parameter element in A a next state. The context should make clear which view is prevalent. But operations of the form $A \to X$ are definitely algebraic (because they gives us information about how to put elements in X), and operations of the form $X \to A$ are coalgebraic (because they give us observable attribute values holding for elements of X). A further complication at this point is that on an initial algebra X one may have operations of the form $X \to A$, obtained by initiality. An example is the length function on lists. Such operations are derived, and are not an integral part of the (definition of the) algebra. Dually, one may have derived operations $A \to X$ on a final coalgebra X.

(2) Algebraic and coalgebraic structures may be found in different hierarchic layers. For example, one can start with certain algebras describing one's application domain. On top of these one can have certain dynamical sys-

tems (processes) as coalgebras, involving such algebras (*e.g.* as codomains of attributes). And such coalgebraic systems may exist in an algebra of processes.

A concrete example of such layering of coalgebra on top of algebra is given by Plotkin's so-called structural operational semantics [Plo81]. It involves a transition system (a coalgebra) describing the operational semantics of some language, by giving the transition rules by induction on the structure of the terms of the language. The latter means that the set of terms of the language is used as (initial) algebra. See Section 1.8 and [RT94, Tur96] for a further investigation of this perspective. Hidden sorted algebras, see [GM94, GD94, BD94, GM96, Mal96] can be seen as other examples: they involve "algebras" with "invisible" sorts, playing a (coalgebraic) role of a state space. Coinduction is used to reason about such hidden state spaces, see [GM96].

1.3 Inductive and coinductive definitions

In the previous section we have seen that "constructor" and "destructor/observer" operations play an important role for algebras and coalgebras, respectively. Constructors tell us how to generate our (algebraic) data elements: the empty list constructor nil and the prefix operation cons generate all finite lists. And destructors (or observers, or transition functions) tell us what we can observe about our data elements: the head and tail operations tell us all about infinite lists: head gives a direct observation, and tail returns a next state.

Once we are aware of this duality between constructing and observing, it is easy to see the difference between inductive and coinductive definitions (relative to given collections of constructors and destructors):

In an *inductive definition* of a function f, one defines the value of f on all constructors.

And:

In a *coinductive definition* of a function f, one defines the values of all destructors on each outcome $f(x)$.

Such a coinductive definition determines the observable behaviour of each $f(x)$.

We shall illustrate inductive and coinductive definitions in some examples involving finite lists (with constructors nil and cons) and infinite lists (with destructors head and tail) over a fixed dataset A, as in the previous section. We assume that inductive definitions are well-known, so we only mention two trivial examples: the (earlier mentioned) function len from finite lists to natural numbers giving the length, and the function $\mathsf{empty?}$ from finite lists to booleans $\{\mathsf{true}, \mathsf{false}\}$ telling whether a list is empty or not:

$$\begin{cases} \mathsf{len}(\mathsf{nil}) = 0 \\ \mathsf{len}(\mathsf{cons}(a, \sigma)) = 1 + \mathsf{len}(\sigma). \end{cases} \qquad \begin{cases} \mathsf{empty?}(\mathsf{nil}) = \mathsf{true} \\ \mathsf{empty?}(\mathsf{cons}(a, \sigma)) = \mathsf{false}. \end{cases}$$

Typically in such inductive definitions, the constructors on the left hand side appear "inside" the function that we are defining. The example of $\mathsf{empty?}$ above, where this does not happen, is a degenerate case.

We turn to examples of coinductive definitions (on infinite lists, say of type A). If we have a function $f \colon A \to A$, then we would like to define an extension $\mathsf{ext}(f)$ of f mapping an infinite list to an infinite list by applying f componentwise. According to the above coinductive definition scheme we have to give the values of the destructors head and tail for a sequence $\mathsf{ext}(f)(\sigma)$. They should be:

$$\begin{cases} \mathsf{head}(\mathsf{ext}(f)(\sigma)) = f(\mathsf{head}(\sigma)) \\ \mathsf{tail}(\mathsf{ext}(f)(\sigma)) = \mathsf{ext}(f)(\mathsf{tail}(\sigma)) \end{cases}$$

Here we clearly see that on the left hand side, the function that we are defining occurs "inside" the destructors. At this stage it is not yet clear if $\mathsf{ext}(f)$ is well-defined, but this is not our concern at the moment.

Alternatively, using the transition relation notation from Example (iv) in the previous section, we can write the definition of $\mathsf{ext}(f)$ as:

$$\frac{\sigma \xrightarrow{\;a\;} \sigma'}{\mathsf{ext}(f)(\sigma) \xrightarrow{\;f(a)\;} \mathsf{ext}(f)(\sigma')}$$

Suppose next, that we wish to define an operation even which takes an infinite list, and produces a new infinite list which contains (in order) all the elements occurring in evenly numbered places of the original list. That is, we would like the operation even to satisfy

$$\mathsf{even}(\sigma(0), \sigma(1), \sigma(2), \ldots) = (\sigma(0), \sigma(2), \sigma(4), \ldots) \tag{1.3}$$

A little thought leads to the following definition clauses.

$$\begin{cases} \mathsf{head}(\mathsf{even}(\sigma)) = \mathsf{head}(\sigma) \\ \mathsf{tail}(\mathsf{even}(\sigma)) = \mathsf{even}(\mathsf{tail}(\mathsf{tail}(\sigma))) \end{cases} \tag{1.4}$$

Or, in the transition relation notation:

$$\frac{\sigma \xrightarrow{\;a\;} \sigma' \xrightarrow{\;a'\;} \sigma''}{\mathsf{even}(\sigma) \xrightarrow{\;a\;} \mathsf{even}(\sigma'')}$$

Let us convince ourselves that this definition gives us what we want. The first clause in (1.4) says that the first element of the list $\mathsf{even}(\sigma)$ is the first element of σ. The next element in $\mathsf{even}(\sigma)$ is $\mathsf{head}(\mathsf{tail}(\mathsf{even}(\sigma)))$, and can be computed as

$$\mathsf{head}(\mathsf{tail}(\mathsf{even}(\sigma))) = \mathsf{head}(\mathsf{even}(\mathsf{tail}(\mathsf{tail}(\sigma)))) = \mathsf{head}(\mathsf{tail}(\mathsf{tail}(\sigma))).$$

Hence the second element in $\mathsf{even}(\sigma)$ is the third element in σ. It is not hard to show for $n \in \mathbb{N}$ that $\mathsf{head}(\mathsf{tail}^{(n)}(\mathsf{even}(\sigma)))$ is the same as $\mathsf{head}(\mathsf{tail}^{(2n)}(\sigma))$.

In a similar way one can coinductively define a function odd which keeps all the oddly listed elements. But it is much easier to define odd as: $\mathsf{odd} = \mathsf{even} \circ \mathsf{tail}$.

As another example, we consider the merge of two infinite lists σ, τ into a single list, by taking elements from σ and τ in turn, starting with σ, say. A coinductive definition of such a function merge requires the outcomes of the destructors head and tail on $\mathsf{merge}(\sigma, \tau)$. They are given as:

$$\begin{cases} \mathsf{head}(\mathsf{merge}(\sigma, \tau)) = \mathsf{head}(\sigma) \\ \mathsf{tail}(\mathsf{merge}(\sigma, \tau)) = \mathsf{merge}(\tau, \mathsf{tail}(\sigma)) \end{cases}$$

In transition system notation, this definition looks as follows.

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

Now one can show that the n-th element of σ occurs as $2n$-th element in $\mathsf{merge}(\sigma, \tau)$, and that the n-th element of τ occurs as $(2n + 1)$-th element of $\mathsf{merge}(\sigma, \tau)$:

$$\begin{aligned} \mathsf{head}(\mathsf{tail}^{(2n)}(\mathsf{merge}(\sigma, \tau))) &= \mathsf{head}(\mathsf{tail}^{(n)}(\sigma)) \\ \mathsf{head}(\mathsf{tail}^{(2n+1)}(\mathsf{merge}(\sigma, \tau))) &= \mathsf{head}(\mathsf{tail}^{(n)}(\tau)). \end{aligned}$$

One can also define a function $\mathsf{merge}_{2,1}(\sigma, \tau)$ which takes two elements of σ for every element of τ (see Exercise 1.10.6).

An obvious result that we would like to prove is: merging the lists of evenly and oddly occuring elements in a list σ returns the original list σ. That is: $\mathsf{merge}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma)) = \sigma$. From what we have seen above we can easily compute that the n-th elements on both sides are equal:

$$\begin{aligned} & \mathsf{head}(\mathsf{tail}^{(n)}(\mathsf{merge}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma)))) \\ = & \begin{cases} \mathsf{head}(\mathsf{tail}^{(m)}(\mathsf{even}(\sigma))) & \text{if } n = 2m \\ \mathsf{head}(\mathsf{tail}^{(m)}(\mathsf{odd}(\sigma))) & \text{if } n = 2m + 1 \end{cases} \\ = & \begin{cases} \mathsf{head}(\mathsf{tail}^{(2m)}(\sigma)) & \text{if } n = 2m \\ \mathsf{head}(\mathsf{tail}^{(2m+1)}(\sigma)) & \text{if } n = 2m + 1 \end{cases} \\ = & \mathsf{head}(\mathsf{tail}^{(n)}(\sigma)). \end{aligned}$$

There is however a more elegant coinductive proof-technique, which will be presented later: in Example 1.6.3 using uniqueness—based on the finality diagram (1.2)—and in the beginning of Section 1.7 using bisimulations.

1.4 Functoriality of products, coproducts and powersets

In the remainder of this paper we shall put the things we have discussed so far in a general framework. Doing so properly requires a certain amount of category theory. We do not intend to describe the relevant matters at the highest level of abstraction, making full use of category theory. Instead, we shall work mainly with ordinary sets. That is, we shall work in the universe given by the category of sets and functions. What we do need is that many operations on sets are "functorial". This means that they do not act only on sets, but also on functions between sets, in an appropriate manner. This is familiar in the computer science literature, not in categorical terminology, but using a "map" terminology. For example, if $\mathsf{list}(A) = A^\star$ describes the set of finite lists of elements of a set A, then for a function $f\colon A \to B$ one can define a function $\mathsf{list}(A) \to \mathsf{list}(B)$ between the corresponding sets of lists, which is usually called[1] $\mathsf{map\_list}(f)$. It sends a finite list $(a_1, \ldots, a_n)$ of elements of A to the list $(f(a_1), \ldots, f(a_n))$ of elements of B, by applying f elementwise. It is not hard to show that this $\mathsf{map\_list}$ operation preserves identity functions and composite functions, *i.e.* that $\mathsf{map\_list}(id_A) = id_{\mathsf{list}(A)}$ and $\mathsf{map\_list}(g \circ f) = \mathsf{map\_list}(g) \circ \mathsf{map\_list}(f)$. This preservation of identities and compositions is the appropriateness that we mentioned above. In this section we concentrate on such functoriality of several basic operations, such as products, coproducts (disjoint unions) and powersets. It will be used in later sections.

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

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

There are then obvious projection functions $\pi\colon X \times Y \to X$ and $\pi'\colon X \times Y \to Y$ by $\pi(x, y) = x$ and $\pi'(x, y) = y$. Also, for functions $f\colon Z \to X$ and $g\colon Z \to Y$ there is a unique "pair function" $\langle f, g\rangle\colon Z \to X \times Y$ with $\pi \circ \langle f, g\rangle = f$ and $\pi' \circ \langle f, g\rangle = g$, namely $\langle f, g\rangle(z) = (f(z), g(z)) \in X \times Y$ for $z \in Z$. Notice that $\langle \pi, \pi'\rangle = id\colon X \times Y \to X \times Y$ and that $\langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle\colon W \to X \times Y$, for functions $h\colon W \to Z$.

Interestingly, the product operation $(X, Y) \mapsto X \times Y$ does not only apply to sets, but also to functions: for functions $f\colon X \to X'$ and $g\colon Y \to Y'$ we can define a function $X \times X' \to Y \times Y'$ by $(x, y) \mapsto (f(x), g(y))$. One writes this

[1] In the category theory literature one uses the same name for the actions of a functor on objects and on morphisms; this leads to the notation $\mathsf{list}(f)$ or $f^\star$ for this function $\mathsf{map\_list}(f)$.

function as $f \times g: X \times Y \to X' \times Y'$, whereby the symbol $\times$ is overloaded: it is used both on sets and on functions. We note that $f \times g$ can be described in terms of projections and pairing as $f \times g = \langle f \circ \pi, g \circ \pi' \rangle$. It is easily verified that the operation $\times$ on functions satisfies

$$id_X \times id_Y = id_{X \times Y} \qquad \text{and} \qquad (f \circ h) \times (g \circ k) = (f \times g) \circ (h \times k).$$

This expresses that the product $\times$ is *functorial*: it does not only apply to sets, but also to functions; and it does so in such a way that identity maps and composites are preserved.

Many more operations are functorial. Also the coproduct (or disjoint union, or sum) $+$ is. For sets X, Y we write their disjoint union as $X + Y$. Explicitly:

$$X + Y = \{\langle 0, x \rangle \mid x \in X\} \cup \{\langle 1, y \rangle \mid y \in Y\}.$$

The first components 0 and 1 serve to force this union to be disjoint. These "tags" enables us to recognise the elements of X and of Y inside $X + Y$. Instead of projections as above we now have "coprojections" $\kappa: X \to X + Y$ and $\kappa': Y \to X + Y$ going in the other direction. One puts $\kappa(x) = \langle 0, x \rangle$ and $\kappa'(y) = \langle 1, y \rangle$. And instead of tupleing we now have "cotupleing" (sometimes called "source tupleing"): for functions $f: X \to Z$ and $g: Y \to Z$ there is a unique function $[f, g]: X + Y \to Z$ with $[f, g] \circ \kappa = f$ and $[f, g] \circ \kappa' = g$. One defines $[f, g]$ by case distinction:

$$[f, g](w) = \begin{cases} f(x) & \text{if } w = \langle 0, x \rangle \\ g(y) & \text{if } w = \langle 1, y \rangle. \end{cases}$$

Notice that $[\kappa, \kappa'] = id$ and $h \circ [f, g] = [h \circ f, h \circ g]$.

This is the coproduct $X + Y$ on sets. We can extend it to functions in the following way. For $f: X \to X'$ and $g: Y \to Y'$ there is a function $f + g: X + Y \to X' + Y'$ by

$$(f + g)(w) = \begin{cases} \langle 0, f(x) \rangle & \text{if } w = \langle 0, x \rangle \\ \langle 1, g(y) \rangle & \text{if } w = \langle 1, y \rangle. \end{cases}$$

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

$$id_X + id_Y = id_{X+Y} \qquad \text{and} \qquad (f \circ h) + (g \circ k) = (f + g) \circ (h + k).$$

We should emphasise that this coproduct $+$ is very different from ordinary union $\cup$. For example, $\cup$ is idempotent: $X \cup X = X$, but there is not odd an isomorphism between $X + X$ and X (if $X \neq \emptyset$).

For a fixed set A, the assignment $X \mapsto X^A = \{f \mid f \text{ is a function } A \to X\}$ is functorial: a function $g: X \to Y$ yields a function $g^A: X^A \to Y^A$ sending $f \in X^A$ to $(g \circ f) \in Y^A$. Clearly, $id^A = id$ and $(h \circ g)^A = h^A \circ g^A$.

Another example of a functorial operation is powerset: $X \mapsto \mathcal{P}(X)$. For a function $f: X \to X'$ one defines $\mathcal{P}(f): \mathcal{P}(X) \to \mathcal{P}(X')$ by

$$U \mapsto \{f(x) \mid x \in U\}.$$

Then $\mathcal{P}(id_X) = id_{\mathcal{P}(X)}$ and $\mathcal{P}(f \circ h) = \mathcal{P}(f) \circ \mathcal{P}(h)$. We shall write $\mathcal{P}_{\text{fin}}(-)$ for the (functorial) operation which maps X to the set of its *finite* subsets.

Here are some trivial examples of functors. The identity operation $X \mapsto X$ is functorial: it acts on functions as $f \mapsto f$. And for a constant set C we have a constant functorial operation $X \mapsto C$; a function $f: X \to X'$ is mapped to the identity function $id_C: C \to C$.

Once we know these actions on functions, we can define functorial operations (or: *functors*, for short) merely by giving their actions on sets. We will often say things like: consider the functor

$$T(X) = X + (C \times X).$$

The action on sets is then $X \mapsto X + (C \times X)$. And for a function $f: X \to X'$ we have an action $T(f)$ of the functor T on f as a function $T(f): T(X) \to T(X')$. Explicitly, $T(f)$ is the function

$$f + (id_C \times f): X + (C \times X) \longrightarrow X' + (C \times X')$$

given by:

$$w \mapsto \begin{cases} \langle 0, f(x) \rangle & \text{if } w = \langle 0, x \rangle \\ \langle 1, (c, f(x)) \rangle & \text{if } w = \langle 1, (c, x) \rangle. \end{cases}$$

The only functors that we shall use in the sequel are such “polynomial” functors T, which are built up with constants, identity functors, products, coproducts and also (finite) powersets. We describe these functors by only giving their actions on sets. Mostly, the functors in this chapter will be of the sort $\mathit{Set} \to \mathit{Set}$, acting on sets and functions between them, with the exception of Section 1.9 on trace semantics where we shall use functors $\mathit{Rel} \to \mathit{Rel}$, acting on sets with relations between them as morphisms.

There is a more general notion of functor $\mathbb{C} \to \mathbb{D}$ as mapping from one “category” $\mathbb{C}$ to another $\mathbb{D}$, see *e.g.* [Awo06]. Here we are only interested in these polynomial functors, going from the category *Set* of sets and functions to itself (or from *Rel* to *Rel*). But much of the theory applies to more general situations.

We shall write $1 = \{*\}$ for a singleton set, with typical inhabitant $*$. Notice that for every set X there is precisely one function $X \to 1$. This says that 1 is *final* (or *terminal*) in the category of sets and functions. And functions $1 \to X$ correspond to elements of X. Usually we shall identify the two. Thus, for example, we sometimes write the empty list as $\mathsf{nil} : 1 \to A^\star$ so that it can be cotupled with the function $\mathsf{cons} : A \times A^\star \to A^\star$ into the algebra

$$[\mathsf{nil}, \mathsf{cons}]: 1 + (A \times A^\star) \to A^\star$$

that will be studied more deeply in Example 1.5.6.

We write 0 for the empty set. For every set X there is precisely one function $0 \to X$, namely the empty function. This property is the *initiality* of 0. These sets 1 and 0 can be seen as the empty product and coproduct.

We list some useful isomorphisms.

$$\begin{array}{rclcrcl}
X \times Y & \cong & Y \times X & \qquad & X + Y & \cong & Y + X \\
1 \times X & \cong & X & & 0 + X & \cong & X \\
X \times (Y \times Z) & \cong & (X \times Y) \times Z & & X + (Y + Z) & \cong & (X + Y) + Z \\
X \times 0 & \cong & 0 & & X \times (Y + Z) & \cong & (X \times Y) + (X \times Z).
\end{array}$$

The last two isomorphisms describe the distribution of products over finite coproducts. We shall often work "up-to" the above isomorphisms, so that we can simply write an n-ary product as $X_1 \times \cdots \times X_n$ without bothering about bracketing.

1.5 Algebras and induction

In this section we start by showing how polynomial functors—as introduced in the previous section—can be used to describe signatures of operations. Algebras of such functors correspond to models of such signatures. They consist of a carrier set with certain functions interpreting the operations. A general notion of homomorphism is defined between such algebras of a functor. This allows us to define initial algebras by the following property: for an arbitrary algebra there is precisely one homomorphism from the initial algebra to this algebra. This turns out to be a powerful notion. It captures algebraic structures which are generated by constructor operations, as will be shown in several examples. Also, it gives rise to the familiar principles of definition by induction and proof by induction.

We start with an example. Let T be the polynomial functor $T(X) = 1 + X + (X \times X)$, and consider for a set U a function $a: T(U) \to U$. Such a map a may be identified with a 3-cotuple $[a_1, a_2, a_3]$ of maps $a_1: 1 \to U$, $a_2: U \to U$ and $a_3: U \times U \to U$ giving us three separate functions going into the set U. They form an example of an algebra (of the functor T): a set together with a (cotupled) number of functions going into that set. For example, if one has a group G, with unit element $e: 1 \to G$, inverse function $i: G \to G$ and multiplication function $m: G \times G \to G$, then one can organise these three maps as an algebra $[e, i, m]: T(G) \to G$ via cotupling[1]. The shape of the functor T determines a certain signature of operations. Had we taken a different functor $S(X) = 1 + (X \times X)$, then maps (algebras of S) $S(U) \to U$ would capture pairs of functions $1 \to U$, $U \times U \to U$ (*e.g.* of a monoid).

Definition 1.5.1 Let T be a functor. An *algebra* of T (or, a T-*algebra*) is a pair consisting of a set U and a function $a: T(U) \to U$.

We shall call the set U the *carrier* of the algebra, and the function a the *algebra structure*, or also the *operation* of the algebra.

For example, the zero and successor functions $0: 1 \to \mathbb{N}$, $S: \mathbb{N} \to \mathbb{N}$ on the natural numbers form an algebra $[0, S]: 1 + \mathbb{N} \to \mathbb{N}$ of the functor $T(X) = 1 + X$. And the set of A-labeled finite binary trees $\mathsf{Tree}(A)$ comes with functions $\mathsf{nil}: 1 \to \mathsf{Tree}(A)$ for the empty tree, and $\mathsf{node}: \mathsf{Tree}(A) \times A \times \mathsf{Tree}(A) \to \mathsf{Tree}(A)$ for constructing a tree out of two (sub)trees and a (node) label. Together, nil and node form an algebra $1 + (\mathsf{Tree}(A) \times A \times \mathsf{Tree}(A)) \to \mathsf{Tree}(A)$ of the functor $S(X) = 1 + (X \times A \times X)$.

We illustrate the link between signatures (of operations) and functors with further details. Let Σ be a (single-sorted, or single-typed) signature, given by a finite collection Σ of operations σ, each with an arity $\mathrm{ar}(\sigma) \in \mathbb{N}$. Each $\sigma \in \Sigma$ will be understood as an operation

$$\sigma: \underbrace{X \times \cdots \times X}_{\mathrm{ar}(\sigma) \text{ times}} \longrightarrow X$$

taking $\mathrm{ar}(\sigma)$ inputs of some type X, and producing an output of type X. With this signature Σ, say with set of operations $\{\sigma_1, \ldots, \sigma_n\}$ we associate a functor

$$T_\Sigma(X) = X^{\mathrm{ar}(\sigma_1)} + \cdots + X^{\mathrm{ar}(\sigma_n)},$$

where for $m \in \mathbb{N}$ the set X^m is the m-fold product $X \times \cdots \times X$. An algebra $a: T_\Sigma(U) \to U$ of this functor T_Σ can be identified with an n-cotuple $a = [a_1, \ldots a_n]: U^{\mathrm{ar}(\sigma_1)} + \cdots + U^{\mathrm{ar}(\sigma_n)} \to U$ of functions $a_i: U^{\mathrm{ar}(\sigma_i)} \to U$ interpreting the operations σ_i in Σ as functions on U. Hence algebras of the functor T_Σ correspond to models of the signature Σ. One sees how the arities in the signature Σ determine the shape of the associated functor T_Σ. Notice that as special case when an arity of an operation is zero we have a constant in Σ. In a T_Σ-algebra $T_\Sigma(U) \to U$ we get an associated map $U^0 = 1 \to U$ giving us an element of the carrier set U as interpretation of the constant. The assumption that the signature Σ is finite is not essential for the correspondence between

[1] Only the group's operations, and not its equations, are captured in this map $T(G) \to G$.

models of Σ and algebras of T_Σ; if Σ is infinite, one can define T_Σ via an infinite coproduct, commonly written as $T_\Sigma(X) = \coprod_{\sigma\in\Sigma} X^{\mathrm{ar}(\sigma)}$.

Polynomial functors T built up from the identity functor, products and coproducts (without constants) have algebras which are models of the kind of signatures Σ described above. This is because by the distribution of products over coproducts one can always write such a functor in "disjunctive normal form" as $T(X) = X^{m_1} + \cdots + X^{m_n}$ for certain natural numbers n and $m_1, \ldots, m_n$. The essential role of the coproducts is to combine multiple operations into a single operation.

The polynomial functors that we use are not only of this form $T(X) = X^{m_1} + \cdots + X^{m_n}$, but may also involve constant sets. This is quite useful, for example, to describe for an arbitrary set A a signature for lists of A's, with function symbols $\mathsf{nil}\colon 1 \to X$ for the empty list, and $\mathsf{cons}\colon A \times X \to X$ for prefixing an element of type A to a list. A model (interpretation) for such a signature is an algebra $T(U) \to U$ of the functor $T(X) = 1 + (A \times X)$ associated with this signature.

We turn to "homomorphisms of algebras", to be understood as structure preserving functions between algebras (of the same signature, or functor). Such a homomorphism is a function between the carrier sets of the algebras which commutes with the operations. For example, suppose we have two algebras $\ell_1\colon 1 \to U_1$, $c_1\colon A \times U_1 \to U_1$ and $\ell_2\colon 1 \to U_2$, $c_2\colon A \times U_2 \to U_2$ of the above list signature. A homomorphism of algebras from the first to the second consists of a function $f\colon U_1 \to U_2$ between the carriers with $f \circ \ell_1 = \ell_2$ and $f \circ c_1 = c_2 \circ (id \times f)$. In two diagrams:

$$\begin{tikzcd} 1 \arrow[r, equal] \arrow[d, "\ell_1"'] & 1 \arrow[d, "\ell_2"] \\ U_1 \arrow[r, "f"'] & U_2 \end{tikzcd} \qquad \text{and} \qquad \begin{tikzcd} A \times U_1 \arrow[r, "id \times f"] \arrow[d, "c_1"'] & A \times U_2 \arrow[d, "c_2"] \\ U_1 \arrow[r, "f"'] & U_2 \end{tikzcd}$$

Thus, writing $n_1 = \ell_1(*)$ and $n_2 = \ell_2(*)$, these diagrams express that $f(n_1) = n_2$ and $f(c_1(a, x)) = c_2(a, f(x))$, for $a \in A$ and $x \in U_1$.

These two diagrams can be combined into a single diagram:

$$\begin{tikzcd} 1 + (A \times U_1) \arrow[r, "id + (id \times f)"] \arrow[d, "{[\ell_1, c_1]}"'] & 1 + (A \times U_2) \arrow[d, "{[\ell_2, c_2]}"] \\ U_1 \arrow[r, "f"'] & U_2 \end{tikzcd}$$

i.e., for the list-functor $T(X) = 1 + (A \times X)$,

$$\begin{tikzcd} T(U_1) \arrow[r, "T(f)"] \arrow[d, "{[\ell_1, c_1]}"'] & T(U_2) \arrow[d, "{[\ell_2, c_2]}"] \\ U_1 \arrow[r, "f"'] & U_2 \end{tikzcd}$$

The latter formulation is entirely in terms of the functor involved. This motivates the following definition.

Definition 1.5.2 Let T be a functor with algebras $a\colon T(U) \to U$ and $b\colon T(V) \to V$. A *homomorphism of algebras* (also called a *map of algebras*, or an *algebra map*) from (U, a) to (V, b) is a function $f\colon U \to V$ between the carrier sets which commutes with the operations: $f \circ a = b \circ T(f)$ in

$$\begin{tikzcd} T(U) \arrow[r, "T(f)"] \arrow[d, "a"'] & T(V) \arrow[d, "b"] \\ U \arrow[r, "f"'] & V \end{tikzcd}$$

As a triviality we notice that for an algebra $a\colon T(U) \to U$ the identity function $U \to U$ is an algebra map $(U, a) \to (U, a)$. And we can compose algebra maps as functions: given two algebra maps

$$\Big(T(U) \xrightarrow{a} U\Big) \xrightarrow{f} \Big(T(V) \xrightarrow{b} V\Big) \xrightarrow{g} \Big(T(W) \xrightarrow{c} W\Big)$$

then the composite function $g \circ f\colon U \to W$ is an algebra map from (U, a) to (W, c). This is because $g \circ f \circ a = g \circ b \circ T(f) = c \circ T(g) \circ T(f) = c \circ T(g \circ f)$, see the following diagram.

$$\begin{tikzcd} T(U) \arrow[r, "T(f)"'] \arrow[rr, bend left, "T(g \circ f)"] \arrow[d, "a"'] & T(V) \arrow[r, "T(g)"'] \arrow[d, "b"'] & T(W) \arrow[d, "c"] \\ U \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & V \arrow[r, "g"] & W \end{tikzcd}$$

Thus: algebras and their homomorphisms form a category.

Now that we have a notion of homomorphism of algebras we can formulate the important concept of "initiality" for algebras.

Definition 1.5.3 An algebra $a: T(U) \to U$ of a functor T is *initial* if for each algebra $b: T(V) \to V$ there is a unique homomorphism of algebras from (U, a) to (V, b). Diagrammatically we express this uniqueness by a dashed arrow, call it f, in

$$\begin{tikzcd} T(U) \arrow[r, dashed, "T(f)"] \arrow[d, "a"'] & T(V) \arrow[d, "b"] \\ U \arrow[r, dashed, "f"'] & V \end{tikzcd}$$

We shall sometimes call this f the "unique mediating algebra map".

We emphasise that unique existence has two aspects, namely *existence* of an algebra map out of the initial algebra to another algebra, and *uniqueness*, in the form of equality of any two algebra maps going out of the initial algebra to some other algebra. Existence will be used as an (inductive) definition principle, and uniqueness as an (inductive) proof principle.

As a first example, we shall describe the set $\mathbb{N}$ of natural numbers as initial algebra.

Example 1.5.4 Consider the set $\mathbb{N}$ of natural number with its zero and successor function $0: 1 \to \mathbb{N}$ and $S: \mathbb{N} \to \mathbb{N}$. These functions can be combined into a single function $[0, S]: 1+\mathbb{N} \to \mathbb{N}$, forming an algebra of the functor $T(X) = 1+X$. We will show that this map $[0, S]: 1+\mathbb{N} \to \mathbb{N}$ is the initial algebra of this functor T. And this characterises the set of natural numbers (up-to-isomorphism), by Lemma 1.5.5 (ii) below.

To prove initiality, assume we have an arbitrary set U carrying a T-algebra structure $[u, h]: 1 + U \to U$. We have to define a "mediating" homomorphism $f: \mathbb{N} \to U$. We try iteration:

$$f(n) = h^{(n)}(u)$$

where we simply write u instead of $u(*)$. That is,

$$f(0) = u \qquad \text{and} \qquad f(n+1) = h(f(n)).$$

These two equations express that we have a commuting diagram

$$\begin{tikzcd} 1+\mathbb{N} \arrow[r, "id+f"] \arrow[d, "{[0,S]}"'] & 1+U \arrow[d, "{[u,h]}"] \\ \mathbb{N} \arrow[r, "f"'] & U \end{tikzcd}$$

making f a homomorphism of algebras. This can be verified easily by distinguishing for an arbitrary element $x \in 1 + \mathbb{N}$ in the upper-left corner the two cases $x = (0, *) = \kappa(*)$ and $x = (1, n) = \kappa'(n)$, for $n \in \mathbb{N}$. In the first case $x = \kappa(*)$ we get

$$f([0, S](\kappa(*))) = f(0) = u = [u, h](\kappa(*)) = [u, h]((id + f)(\kappa(*))).$$

In the second case $x = \kappa'(n)$ we similarly check:

$$f([0, S](\kappa'(n))) = f(S(n)) = h(f(n)) = [u, h](\kappa'(f(n))) = [u, h]((id+f)(\kappa'(n))).$$

Hence we may conclude that $f([0, S](x)) = [u, h]((id + f)(x))$, for all $x \in 1 + \mathbb{N}$, *i.e.* that $f \circ [0, S] = [u, h] \circ (id + f)$.

This looks promising, but we still have to show that f is the only map making the diagram commute. If $g\colon \mathbb{N} \to U$ also satisfies $g \circ [0, S] = [u, h] \circ (id + g)$, then $g(0) = u$ and $g(n + 1) = h(g(n))$, by the same line of reasoning followed above. Hence $g(n) = f(n)$ by induction on n, so that $g = f\colon \mathbb{N} \to U$.

We shall give a simple example showing how to use this initiality for inductive definitions. Suppose we wish to define by induction the function $f(n) = 2^{-n}$ from the natural numbers $\mathbb{N}$ to the rational numbers $\mathbb{Q}$. Its defining equations are:

$$f(0) = 1 \qquad \text{and} \qquad f(n + 1) = \tfrac{1}{2} f(n).$$

In order to define this function $f\colon \mathbb{N} \to \mathbb{Q}$ by initiality, we have to put an algebra structure $1 + \mathbb{Q} \to \mathbb{Q}$ on the set of rational numbers $\mathbb{Q}$, see the above definition. This algebra on $\mathbb{Q}$ corresponds to the right hand side of the two defining equations of f, given as two functions

$$\begin{array}{ccc} 1 \xrightarrow{\;1\;} \mathbb{Q} & \qquad\qquad & \mathbb{Q} \xrightarrow{\;\frac{1}{2}(-)\;} \mathbb{Q} \\ * \longmapsto 1 & & x \longmapsto \frac{1}{2}x \end{array}$$

(where we use '1' both for the singleton set $1 = \{*\}$ and for the number $1 \in \mathbb{Q}$) which combine into a single function

$$1 + \mathbb{Q} \xrightarrow{\;[1, \frac{1}{2}(-)]\;} \mathbb{Q}$$

forming an algebra on $\mathbb{Q}$. The function $f(n) = 2^{-n}$ is then determined by initiality as the unique function making the following diagram commute.

$$\begin{tikzcd}
1+\mathbb{N} \arrow[r, "id+f"] \arrow[d, "{[0,S]}"'] & 1+\mathbb{Q} \arrow[d, "{[1,\frac{1}{2}(-)]}"] \\
\mathbb{N} \arrow[r, "f"'] & \mathbb{Q}
\end{tikzcd}$$

This shows how initiality can be used to define functions by induction. It requires that one puts an appropriate algebra structure on the codomain (*i.e.* the range) of the intended function, corresponding to the induction clauses that determine the function.

We emphasise that the functor T is a parameter in Definitions 1.5.2 and 1.5.3 of "homomorphism" and "initiality" for algebras, yielding uniform notions for all functors T (representing certain signatures). It turns out that initial algebras have certain properties, which can be shown for all functors T at once. Diagrams are convenient in expressing and proving these properties, because they display information in a succinct way. And they are useful both in existence and uniqueness arguments.

Lemma 1.5.5 Let T be a functor.

(i) Initial T-algebras, if they exist, are unique, up-to-isomorphism of algebras. That is, if we have two initial algebras $a: T(U) \to U$ and $a': T(U') \to U'$ of T, then there is a unique isomorphism $f: U \xrightarrow{\cong} U'$ of algebras:

$$\begin{tikzcd}
T(U) \arrow[r, "T(f)", "\cong"'] \arrow[d, "a"'] & T(U') \arrow[d, "a'"] \\
U \arrow[r, "f", "\cong"'] & U'
\end{tikzcd}$$

(ii) The operation of an initial algebras is an isomorphism: if $a: T(U) \to U$ is initial algebra, then a has an inverse $a^{-1}: U \to T(U)$.

The first point tells us that a functor can have (essentially) at most one initial algebra[1]. Therefore, we often speak of *the* initial algebra of a functor T. And the second point—which is due to Lambek—says that an initial algebra $T(U) \to U$ is a fixed point $T(U) \cong U$ of the functor T. Initial algebras may be seen as generalizations of least fixed points of monotone functions, since they have a (unique) map into an arbitrary algebra, see Exercise 1.10.3.

[1] This is a more general property of initial objects in a category.

Proof (i) Suppose both $a: T(U) \to U$ and $a': T(U') \to U'$ are initial algebras of the functor T. By initiality of a there is a unique algebra map $f: U \to U'$. Similarly, by initiality of a' there is a unique algebra map $f': U' \to U$ in the other direction:

$$\begin{tikzcd} T(U) \arrow[r, dashed, "T(f)"] \arrow[d, "a"'] & T(U') \arrow[d, "a'"] \\ U \arrow[r, dashed, "f"'] & U' \end{tikzcd} \qquad \begin{tikzcd} T(U') \arrow[r, dashed, "T(f')"] \arrow[d, "a'"'] & T(U) \arrow[d, "a"] \\ U' \arrow[r, dashed, "f'"'] & U \end{tikzcd}$$

Here we use the existence parts of initiality. The uniqueness part gives us that the two resulting algebra maps $(U, a) \to (U, a)$, namely $f \circ f'$ and id in:

$$\begin{tikzcd} T(U) \arrow[r, "T(f)"] \arrow[d, "a"'] & T(U') \arrow[r, "T(f')"] \arrow[d, "a'"'] & T(U) \arrow[d, "a"] \\ U \arrow[r, "f"'] & U' \arrow[r, "f'"'] & U \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} T(U) \arrow[r, "T(id)"] \arrow[d, "a"'] & T(U) \arrow[d, "a"] \\ U \arrow[r, "id"'] & U \end{tikzcd}$$

must be equal, *i.e.* that $f' \circ f = id$. Uniqueness of algebra maps $(U', a') \to (U', a')$ similarly yields $f \circ f' = id$. Hence f is an isomorphism of algebras.

(ii) Let $a: T(U) \to U$ be initial T-algebra. In order to show that the function a is an isomorphism, we have to produce an inverse function $U \to T(U)$. Initiality of (U, a) can be used to define functions out of U to arbitrary algebras. Since we seek a function $U \to T(U)$, we have to put an algebra structure on the set $T(U)$. A moment's thought yields a candidate, namely the result $T(a): T(T(U)) \to T(U)$ of applying the functor T to the function a. This function $T(a)$ gives by initiality of $a: T(U) \to U$ rise to a function $a': U \to T(U)$ with $T(a) \circ T(a') = a' \circ a$ in:

$$\begin{tikzcd} T(U) \arrow[r, dashed, "T(a')"] \arrow[d, "a"'] & T(T(U)) \arrow[d, "T(a)"] \\ U \arrow[r, dashed, "a'"'] & T(U) \end{tikzcd}$$

The function $a \circ a' \colon U \to U$ is an algebra map $(U, a) \to (U, a)$:

$$\begin{tikzcd} T(U) \arrow[r, "T(a')"] \arrow[d, "a"'] & T(T(U)) \arrow[r, "T(a)"] \arrow[d, "T(a)"'] & T(U) \arrow[d, "a"] \\ U \arrow[r, "a'"'] & T(U) \arrow[r, "a"'] & U \end{tikzcd}$$

so that $a \circ a' = id$ by uniqueness of algebra maps $(U, a) \to (U, a)$. But then

$$\begin{array}{rcll} a' \circ a & = & T(a) \circ T(a') & \text{by definition of } a' \\ & = & T(a \circ a') & \text{since } T \text{ preserves composition} \\ & = & T(id) & \text{as we have just seen} \\ & = & id & \text{since } T \text{ preserves identities.} \end{array}$$

Hence $a \colon T(U) \to U$ is an isomorphism with a' as its inverse. □

From now on we shall often write an initial T-algebra as a map $a \colon T(U) \xrightarrow{\cong} U$, making this isomorphism explicit.

Example 1.5.6 Let A be fixed set and consider the functor $T(X) = 1+(A\times X)$ that we used earlier to capture models of the list signature $1 \to X$, $A \times X \to X$. We claim that the initial algebra of this functor T is the set $A^\star = \mathsf{list}(A) = \bigcup_{n\in\mathbb{N}} A^n$ of finite sequences of elements of A, together with the function (or element) $1 \to A^\star$ given by the empty list $\mathsf{nil} = ()$, and the function $A \times A^\star \to A^\star$ which maps an element $a \in A$ and a list $\alpha = (a_1, \ldots, a_n) \in A^\star$ to the list $\mathsf{cons}(a, \alpha) = (a, a_1, \ldots, a_n) \in A^\star$, obtained by prefixing a to α. These two functions can be combined into a single function $[\mathsf{nil}, \mathsf{cons}] \colon 1 + (A \times A^\star) \to A^\star$, which, as one easily checks, is an isomorphism. But this does not yet mean that it is the initial algebra. We will check this explicitly.

For an arbitrary algebra $[u, h] \colon 1 + (A \times U) \to U$ of the list-functor T we have a unique homomorphism $f \colon A^\star \to U$ of algebras:

$$\begin{tikzcd} 1 + (A \times A^\star) \arrow[r, "id + (id \times f)"] \arrow[d, "{[\mathsf{nil}, \mathsf{cons}]}"'] & 1 + (A \times U) \arrow[d, "{[u, h]}"] \\ A^\star \arrow[r, "f"'] & U \end{tikzcd}$$

namely

$$f(\alpha) = \begin{cases} u & \text{if } \alpha = \mathsf{nil} \\ h(a, f(\beta)) & \text{if } \alpha = \mathsf{cons}(a, \beta). \end{cases}$$

We leave it to the reader to verify that f is indeed the unique function $A^\star \to U$ making the diagram commute.

Again we can use this initiality of $A^\star$ to define functions by induction (for lists). As example we take the length function $\mathsf{len}\colon A^\star \to \mathbb{N}$, described already in the beginning of Section 1.2. In order to define it by initiality, it has to arise from a list-algebra structure $1 + A \times \mathbb{N} \to \mathbb{N}$ on the natural numbers $\mathbb{N}$. This algebra structure is the cotuple of the two functions $0\colon 1 \to \mathbb{N}$ and $S \circ \pi'\colon A \times \mathbb{N} \to \mathbb{N}$. Hence len is determined as the unique function in the following initiality diagram.

$$\begin{tikzcd}
1 + (A \times A^\star) \arrow[r, "id + (id \times \mathsf{len})"] \arrow[d, "{[\mathsf{nil}, \mathsf{cons}]}"', "\cong"] & 1 + (A \times \mathbb{N}) \arrow[d, "{[0, S \circ \pi']}"] \\
A^\star \arrow[r, "\mathsf{len}"'] & \mathbb{N}
\end{tikzcd}$$

The algebra structure that we use on $\mathbb{N}$ corresponds to the defining clauses $\mathsf{len}(\mathsf{nil}) = 0$ and $\mathsf{len}(\mathsf{cons}(a, \alpha)) = S(\mathsf{len}(\alpha)) = S(\mathsf{len}(\pi'(a, \alpha))) = S(\pi'(id \times \mathsf{len})(a, \alpha))$.

We proceed with an example showing how proof by induction involves using the uniqueness of a map out of an initial algebra. Consider therefore the "doubling" function $d\colon A^\star \to A^\star$ which replaces each element a in a list α by two consecutive occurrences a, a in $d(\alpha)$. This function is defined as the unique one making the following diagram commute.

$$\begin{tikzcd}
1 + (A \times A^\star) \arrow[r, "id + (id \times d)"] \arrow[d, "{[\mathsf{nil}, \mathsf{cons}]}"', "\cong"] & 1 + (A \times A^\star) \arrow[d, "{[\mathsf{nil}, \lambda(a, \alpha).\ \mathsf{cons}(a, \mathsf{cons}(a, \alpha))]}"] \\
A^\star \arrow[r, "d"'] & A^\star
\end{tikzcd}$$

That is, d is defined by the induction clauses $d(\mathsf{nil}) = \mathsf{nil}$ and $d(\mathsf{cons}(a, \alpha)) = \mathsf{cons}(a, \mathsf{cons}(a, d(\alpha))$. We wish to show that the length of the list $d(\alpha)$ is twice the length of α, *i.e.* that

$$\mathsf{len}(d(\alpha)) = 2 \cdot \mathsf{len}(\alpha).$$

The ordinary induction proof consists of two steps:

$$\mathsf{len}(d(\mathsf{nil})) = \mathsf{len}(\mathsf{nil}) = 0 = 2 \cdot 0 = 2 \cdot \mathsf{len}(\mathsf{nil})$$

And

$$
\begin{aligned}
\mathsf{len}(d(\mathsf{cons}(a,\alpha))) &= \mathsf{len}(\mathsf{cons}(a,\mathsf{cons}(a,d(\alpha)))) \\
&= 1+1+\mathsf{len}(d(\alpha)) \\
&\overset{\text{(IH)}}{=} 2+2\cdot\mathsf{len}(\alpha) \\
&= 2\cdot(1+\mathsf{len}(\alpha)) \\
&= 2\cdot\mathsf{len}(\mathsf{cons}(a,\alpha)).
\end{aligned}
$$

The "initiality" induction proof of the fact $\mathsf{len}\circ d = 2\cdot(-)\circ\mathsf{len}$ uses uniqueness in the following manner. Both $\mathsf{len}\circ d$ and $2\cdot(-)\circ\mathsf{len}$ are homomorphism from the (initial) algebra $(A^\star, [\mathsf{nil},\mathsf{cons}])$ to the algebra $(\mathbb{N}, [0, S\circ S\circ\pi'])$, so they must be equal by initiality. First we check that $\mathsf{len}\circ d$ is an appropriate homomorphism by inspection of the following diagram.

$$
\begin{tikzcd}
1+(A\times A^\star) \arrow[r, "id+(id\times d)"] \arrow[d, "\cong"', "{[\mathsf{nil},\mathsf{cons}]}"] & 1+(A\times A^\star) \arrow[r, "id+(id\times\mathsf{len})"] \arrow[d, "{[\mathsf{nil},\,\lambda(a,\alpha).\ \mathsf{cons}(a,\mathsf{cons}(a,\alpha))]}"] & 1+(A\times\mathbb{N}) \arrow[d, "{[0,S\circ S\circ\pi']}"] \\
A^\star \arrow[r, "d"'] & A^\star \arrow[r, "\mathsf{len}"'] & \mathbb{N}
\end{tikzcd}
$$

The rectangle on the left commutes by definition of d. And commutation of the rectangle on the right follows easily from the definition of len. Next we check that $2\cdot(-)\circ\mathsf{len}$ is also a homomorphism of algebras:

$$
\begin{tikzcd}
1+(A\times A^\star) \arrow[r, "id+(id\times\mathsf{len})"] \arrow[dd, "\cong"', "{[\mathsf{nil},\mathsf{cons}]}"] & 1+(A\times\mathbb{N}) \arrow[r, "id+(id\times 2\cdot(-))"] \arrow[d, "id+\pi'"'] & 1+(A\times\mathbb{N}) \arrow[d, "id+\pi'"'] \arrow[dd, bend left=60, "{[0,S\circ S\circ\pi']}"] \\
& 1+\mathbb{N} \arrow[r, "id+2\cdot(-)"] \arrow[d, "{[0,S]}"', "\cong"] & 1+\mathbb{N} \arrow[d, "{[0,S\circ S]}"'] \\
A^\star \arrow[r, "\mathsf{len}"'] & \mathbb{N} \arrow[r, "2\cdot(-)"'] & \mathbb{N}
\end{tikzcd}
$$

The square on the left commutes by definition of len. Commutation of the upper square on the right follows from an easy computation. And the lower square on the right may be seen as defining the function $2\cdot(-)\colon\mathbb{N}\to\mathbb{N}$ by the clauses: $2\cdot 0 = 0$ and $2\cdot(S(n)) = S(S(2\cdot n))$—which we took for granted in the earlier "ordinary" proof.

We conclude our brief discussion of algebras and induction with a few remarks.

(1) Given a number of constructors one can form the carrier set of the associated initial algebra as the set of 'closed' terms (or 'ground' terms, not

containing variables) that can be formed with these constructors. For example, the zero and successor constructors $\mathsf{0}\colon 1 \to X$ and $\mathsf{S}\colon X \to X$ give rise to the set of closed terms,

$$\{\mathsf{0}, \mathsf{S}(\mathsf{0}), \mathsf{S}(\mathsf{S}(\mathsf{0}))), \ldots\}$$

which is (isomorphic to) the set $\mathbb{N}$ of natural numbers. Similarly, the set of closed terms arising from the A-list constructors $\mathsf{nil}\colon 1 \to X$, $\mathsf{cons}\colon A \times X \to X$ is the set $A^\star$ of finite sequences (of elements of A).

Although it is pleasant to know what an initial algebra looks like, in using initiality we do not need this knowledge. All we need to know is that there exists an initial algebra. Its defining property is sufficient to use it. There are abstract results, guaranteeing the existence of initial algebras for certain (continuous) functors, see *e.g.* [LS81, SP82], where initial algebras are constructed as suitable colimits, generalizing the construction of least fixed points of continuous functions.

(2) The initiality format of induction has the important advantage that it generalises smoothly from natural numbers to other (algebraic) data types, like lists or trees. Once we know the signature containing the constructor operations of these data types, we know what the associated functor is and we can determine its initial algebra. This uniformity provided by initiality was first stressed by the "ADT-group" [GTW78], and forms the basis for inductively defined types in many programming languages. For example, in the (functional) language ML, the user can introduce a new inductive type X via the notation

$$\mathsf{datatype}\ X = c_1\ \mathsf{of}\ \sigma_1(X)\ \mid\ \cdots\ \mid\ c_n\ \mathsf{of}\ \sigma_n(X).$$

The idea is that X is the carrier of the initial algebra associated with the constructors $c_1\colon \sigma_1(X) \to X, \ldots, c_n\colon \sigma_n(X) \to X$. That is, with the functor $T(X) = \sigma_1(X) + \cdots + \sigma_n(X)$. The σ_i are existing types which may contain X (positively)[1]. The uniformity provided by the initial algebra format (and dually also by the final coalgebra format) is very useful if one wishes to automatically generate various rules associated with (co)inductively defined types (for example in programming languages like CHARITY [CS95] or in proof tools like PVS [ORR$^+$96], HOL/ISABELLE [GM93, Mel89, Pau90, Pau97], or COQ [PM93]).

Another advantage of the initial algebra format is that it is dual to the

[1] This definition scheme in ML contains various aspects which are not investigated here, *e.g.* it allows (a) $X = X(\vec{\alpha})$ to contain type variables $\vec{\alpha}$, (b) mutual dependencies between such definitions, (c) iteration of inductive definitions (so that, for example, the LIST operation which is obtained via this scheme can be used in the σ_i.

final coalgebra format, as we shall see in the next section. This forms the basis for the duality between induction and coinduction.

(3) We have indicated only in one example that uniqueness of maps out of an initial algebra corresponds to proof (as opposed to definition) by induction. To substantiate this claim further we show how the usual predicate formulation of induction for lists can be derived from the initial algebra formulation. This predicate formulation says that a predicate (or subset) $P \subseteq A^\star$ is equal to $A^\star$ in case $\mathsf{nil} \in P$ and $\alpha \in P \Rightarrow \mathsf{cons}(a, \alpha) \in P$, for all $a \in A$ and $\alpha \in A^\star$. Let us consider P as a set in its own right, with an explicit inclusion function $i\colon P \to A^\star$ (given by $i(x) = x$). The induction assumptions on P essentially say that P carries an algebra structure $\mathsf{nil}\colon 1 \to P$, $\mathsf{cons}\colon A \times P \to P$, in such a way that the inclusion map $i\colon P \to A^\star$ is a map of algebras:

$$\begin{tikzcd}
1 + (A \times P) \arrow[r, "id + (id \times i)"] \arrow[d, "{[\mathsf{nil}, \mathsf{cons}]}"'] & 1 + (A \times A^\star) \arrow[d, "{[\mathsf{nil}, \mathsf{cons}]}", "\cong"'] \\
P \arrow[r, "i"'] & A^\star
\end{tikzcd}$$

In other words: P is a subalgebra of $A^\star$. By initiality we get a function $j\colon A^\star \to P$ as on the left below. But then $i \circ j = id$, by uniqueness.

$$\begin{tikzcd}
1 + (A \times A^\star) \arrow[r, "id + (id \times j)"'] \arrow[rr, bend left, "id + (id \times id)"] \arrow[d, "{[\mathsf{nil}, \mathsf{cons}]}"', "\cong"] & 1 + (A \times P) \arrow[r, "id + (id \times i)"'] \arrow[d, "{[\mathsf{nil}, \mathsf{cons}]}"] & 1 + (A \times A^\star) \arrow[d, "{[\mathsf{nil}, \mathsf{cons}]}", "\cong"'] \\
A^\star \arrow[r, "j"] \arrow[rr, bend right, "id"'] & P \arrow[r, "i"] & A^\star
\end{tikzcd}$$

This means that $P = A^\star$, as we wished to derive.

(4) The initiality property from Definition 1.5.3 allows us to define functions $f\colon U \to V$ out of an initial algebra (with carrier) U. Often one wishes to define functions $U \times D \to V$ involving an additional parameter ranging over a set D. A typical example is the addition function $\mathsf{plus}\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, defined by induction on (say) its first argument, with the second argument as parameter. One can handle such functions $U \times D \to V$ via Currying: they correspond to functions $U \to V^D$. And the latter can be defined via the initiality scheme. For example, we can define a Curryied addition function $\mathsf{plus}\colon \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$ via initiality by putting an appropriate algebra

structure $1 + \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ on $\mathbb{N}$ (see Example 1.5.4):

$$\begin{tikzcd}
1 + \mathbb{N} \arrow[r, "\mathsf{plus}"] \arrow[d, "{[0,S]}"', "\cong"] & 1 + \mathbb{N}^{\mathbb{N}} \arrow[d, "{[\lambda x.\, x,\ \lambda f.\, \lambda x.\, S(f(x))]}"] \\
\mathbb{N} \arrow[r, "\mathsf{plus}"'] & \mathbb{N}^{\mathbb{N}}
\end{tikzcd}$$

This says that

$$\mathsf{plus}(0) = \lambda x.\, x \qquad \text{and} \qquad \mathsf{plus}(n+1) = \lambda x.\, S(\mathsf{plus}(n)(x)).$$

Alternatively, one may formulate initiality "with parameters", see [Jac95], so that one can handle such functions $U \times D \to V$ directly.

1.6 Coalgebras and coinduction

In Section 1.4 we have seen that a "co"-product $+$ behaves like a product $\times$, except that the arrows point in opposite direction: one has coprojections $X \to X + Y \leftarrow Y$ instead of projections $X \leftarrow X \times Y \to Y$, and cotupleing instead of tupleing. One says that the coproduct $+$ is the dual of the product $\times$, because the associated arrows are reversed. Similarly, a "co"-algebra is the dual of an algebra.

Definition 1.6.1 For a functor T, a *coalgebra* (or a T-*coalgebra*) is a pair (U, c) consisting of a set U and a function $c\colon U \to T(U)$.

Like for algebras, we call the set U the *carrier* and the function c the *structure* or *operation* of the coalgebra (U, c). Because coalgebras often describe dynamical systems (of some sort), the carrier set U is also called the *state space*.

What, then, is the difference between an algebra $T(U) \to U$ and a coalgebra $U \to T(U)$? Essentially, it is the difference between construction and observation. An algebra consists of a carrier set U with a function $T(U) \to U$ going *into* this carrier U. It tells us how to construct elements in U. And a coalgebra consists of a carrier set U with a function $U \to T(U)$ in the opposite direction, going *out of* U. In this case we do not know how to form elements in U, but we only have operations acting on U, which may give us some information about U. In general, these coalgebraic operations do not tell us all there is to say about elements of U, so that we only have *limited access* to U. Coalgebras—like algebras—can be seen as models of a signature of operations—not of constructor operations, but of destructor/observer operations.

Consider for example the functor $T(X) = A \times X$, where A is a fixed set. A coalgebra $U \to T(U)$ consists of two functions $U \to A$ and $U \to U$, which we earlier called $\mathsf{value}: U \to A$ and $\mathsf{next}: U \to U$. With these operations we can do two things, given an element $u \in U$:

(1) produce an element in A, namely $\mathsf{value}(u)$;
(2) produce a next element in U, namely $\mathsf{next}(u)$.

Now we can repeat (1) and (2) and form another element in A, namely $\mathsf{value}(\mathsf{next}(u))$. By proceeding in this way we can get for each element $u \in U$ an infinite sequence $(a_1, a_2, \ldots) \in A^{\mathbb{N}}$ of elements $a_n = \mathsf{value}(\mathsf{next}^{(n)}(u)) \in A$. This sequence of elements that u gives rise to is what we can *observe* about u. Two elements $u_1, u_2 \in U$ may well give rise to the same sequence of elements of A, without actually being equal as elements of U. In such a case one calls u_1 and u_2 observationally indistinguishable, or bisimilar.

Here is another example. Let the functor $T(X) = 1 + A \times X$ have a coalgebra $\mathsf{pn}: U \to 1 + A \times U$, where '$\mathsf{pn}$' stands for 'possible next'. If we have an element $u \in U$, then we can see the following.

(1) Either $\mathsf{pn}(u) = \kappa(*) \in 1 + A \times U$ is in the left component of $+$. If this happens, then our experiment stops, since there is no state (element of U) left with which to continue.
(2) Or $\mathsf{pn}(u) = \kappa'(a, u) \in 1 + A \times U$ is in the right $+$-component. This gives us an element $a \in A$ and a next element $u' \in U$ of the carrier, with which we can proceed.

Repeating this we can observe for an element $u \in U$ either a finite sequence $(a_1, a_2, \ldots, a_n) \in A^\star$ of, or an infinite sequence $(a_1, a_2, \ldots) \in A^{\mathbb{N}}$. The observable outcomes are elements of the set $A^\infty = A^\star + A^{\mathbb{N}}$ of finite and infinite lists of a's.

These observations will turn out to be elements of the final coalgebra of the functors involved, see Example 1.6.3 and 1.6.5 below. But in order to formulate this notion of finality for coalgebras we first need to know what a "homomorphism of coalgebras" is. It is, like in algebra, a function between the underlying sets which commutes with the operations. For example, let $T(X) = A \times X$ be the "infinite list" functor as used above, with coalgebras $\langle h_1, t_1 \rangle: U_1 \to A \times U_1$ and $\langle h_2, t_2 \rangle: U_2 \to A \times U_2$. A homomorphism of coalgebras from the first to the second consists of a function $f: U_1 \to U_2$ between the carrier sets (state spaces) with $h_2 \circ f = h_1$ and $t_2 \circ f = f \circ t_1$ in:

$$\begin{tikzcd} U_1 \arrow[r, "f"] \arrow[d, "h_1"'] & U_2 \arrow[d, "h_2"] \\ A \arrow[r, equal] & A \end{tikzcd} \qquad \text{and} \qquad \begin{tikzcd} U_1 \arrow[r, "f"] \arrow[d, "t_1"'] & U_2 \arrow[d, "t_2"] \\ U_1 \arrow[r, "f"'] & U_2 \end{tikzcd}$$

These two diagrams can be combined into a single one:

$$\begin{tikzcd} U_1 \arrow[r, "f"] \arrow[d, "\langle h_1, t_1 \rangle"'] & U_2 \arrow[d, "\langle h_2, t_2 \rangle"] \\ A \times U_1 \arrow[r, "id \times f"'] & A \times U_2 \end{tikzcd}$$

that is, into

$$\begin{tikzcd} U_1 \arrow[r, "f"] \arrow[d, "\langle h_1, t_1 \rangle"'] & U_2 \arrow[d, "\langle h_2, t_2 \rangle"] \\ T(U_1) \arrow[r, "T(f)"'] & T(U_2) \end{tikzcd}$$

Definition 1.6.2 Let T be a functor.

(i) A *homomorphism of coalgebras* (or, *map of coalgebras*, or *coalgebra map*) from a T-coalgebra $U_1 \xrightarrow{c_1} T(U_1)$ to another T-coalgebra $U_2 \xrightarrow{c_2} T(U_2)$ consists of a function $f: U_1 \to U_2$ between the carrier sets which commutes with the operations: $c_2 \circ f = T(f) \circ c_1$ as expressed by the following diagram.

$$\begin{tikzcd} U_1 \arrow[r, "f"] \arrow[d, "c_1"'] & U_2 \arrow[d, "c_2"] \\ T(U_1) \arrow[r, "T(f)"'] & T(U_2) \end{tikzcd}$$

(ii) A *final coalgebra* $d: Z \to T(Z)$ is a coalgebra such that for every coalgebra $c: U \to T(U)$ there is a unique map of coalgebras $(U, c) \to (Z, d)$.

Notice that where the initiality property for algebras allows us to define functions going *out of* an initial algebra, the finality property for coalgebras gives us means to define functions *into* a final coalgebra. Earlier we have emphasised that what is typical in a coalgebraic setting is that there are no operations for constructing elements of a state space (of a coalgebra), and that state spaces should therefore be seen as black boxes. However, if we know that a certain coalgebra is final, then we can actually form elements in its state space by this finality principle. The next example contains some illustrations. Besides a means for constructing elements, finality also allows us to define various operations on final coalgebras, as will be shown in a series of examples below. In fact, in this way one can put certain algebraic structure on top of a coalgebra, see [Tur96] for a systematic study in the context of process algebras.

Now that we have seen the definitions of initiality (for algebras, see Definition 1.5.3) and finality (for coalgebras) we are in a position to see their similarities. At an informal level we can explain these similarities as follows. A typical initiality diagram may be drawn as:

$$\begin{tikzcd} T(U) \arrow[r, dashed] \arrow[d, "\cong", "\text{initial algebra}"'] & T(V) \arrow[d, "\text{base step plus next step}"] \\ U \arrow[r, dashed, "\text{``and-so-forth''}"'] & V \end{tikzcd}$$

The map "and-so-forth" that is defined in this diagram applies the "next step" operations repeatedly to the "base step". The pattern in a finality diagram is similar:

$$\begin{tikzcd} V \arrow[r, dashed, "\text{``and-so-forth''}"] \arrow[d, "\text{observe plus next step}"'] & U \arrow[d, "\cong"', "\text{final coalgebra}"] \\ T(V) \arrow[r, dashed] & T(U) \end{tikzcd}$$

In this case the "and-so-forth" map captures the observations that arise by repeatedly applying the "next step" operation. This captures the observable behaviour.

The technique for defining a function $f: V \to U$ by finality is thus: describe the direct observations together with the single next steps of f as a coalgebra structure on V. The function f then arises by repetition. Hence a coinductive definition of f does not determine f "at once", but "step-by-step". In the next section we shall describe proof techniques using bisimulations, which fully exploit this step-by-step character of coinductive definitions.

But first we identify a simply coalgebra concretely, and show how we can use finality.

Example 1.6.3 For a fixed set A, consider the functor $T(X) = A \times X$. We claim that the final coalgebra of this functor is the set $A^{\mathbb{N}}$ of infinite lists of elements from A, with coalgebra structure

$$\langle \mathsf{head}, \mathsf{tail} \rangle : A^{\mathbb{N}} \longrightarrow A \times A^{\mathbb{N}}$$

given by

$$\mathsf{head}(\alpha) = \alpha(0) \qquad \text{and} \qquad \mathsf{tail}(\alpha) = \lambda x.\ \alpha(x+1).$$

Hence head takes the first element of an infinite sequence $(\alpha(0), \alpha(1), \alpha(2), \ldots)$

of elements of A, and tail takes the remaining list. We notice that the pair of functions $\langle \mathsf{head}, \mathsf{tail} \rangle : A^{\mathbb{N}} \to A \times A^{\mathbb{N}}$ is an isomorphism.

We claim that for an arbitrary coalgebra $\langle \mathsf{value}, \mathsf{next} \rangle : U \to A \times U$ there is a unique homomorphism of coalgebras $f : U \to A^{\mathbb{N}}$; it is given for $u \in U$ and $n \in \mathbb{N}$ by

$$f(u)(n) = \mathsf{value}\left(\mathsf{next}^{(n)}(u)\right).$$

Then indeed, $\mathsf{head} \circ f = \mathsf{value}$ and $\mathsf{tail} \circ f = f \circ \mathsf{next}$, making f a map of coalgebras. And f is unique in satisfying these two equations, as can be checked easily.

Earlier in this section we saw that what we can observe about an element $u \in U$ is an infinite list of elements of A arising as $\mathsf{value}(u)$, $\mathsf{value}(\mathsf{next}(u))$, $\mathsf{value}(\mathsf{next}(\mathsf{next}(u))), \ldots$ Now we see that this observable behaviour of u is precisely the outcome $f(u) \in A^{\mathbb{N}}$ at u of the unique map f to the final coalgebra. Hence the elements of the final coalgebra give the observable behaviour. This is typical for final coalgebras.

Once we know that $A^{\mathbb{N}}$ is a final coalgebra—or, more precisely, carries a final coalgebra structure—we can use this finality to define functions into $A^{\mathbb{N}}$. Let us start with a simple example, which involves defining the constant sequence $\mathsf{const}(a) = (a, a, a, \ldots) \in A^{\mathbb{N}}$ by coinduction (for some element $a \in A$). We shall define this constant as a function $\mathsf{const}(a) : 1 \to A^{\mathbb{N}}$, where $1 = \{*\}$ is a singleton set. Following the above explanation, we have to produce a coalgebra structure $1 \to T(1) = A \times 1$ on 1, in such a way that $\mathsf{const}(a)$ arises by repetition. In this case the only thing we want to observe is the element $a \in A$ itself, and so we simply define as coalgebra structure $1 \to A \times 1$ the function $* \mapsto (a, *)$. Indeed, $\mathsf{const}(a)$ arises in the following finality diagram.

$$\begin{tikzcd}
1 \arrow[r, "\mathsf{const}(a)"] \arrow[d, "* \mapsto (a,*)"'] & A^{\mathbb{N}} \arrow[d, "\cong"', "\langle \mathsf{head}, \mathsf{tail} \rangle"] \\
A \times 1 \arrow[r, "id \times \mathsf{const}(a)"'] & A \times A^{\mathbb{N}}
\end{tikzcd}$$

It expresses that $\mathsf{head}(\mathsf{const}(a)) = a$ and $\mathsf{tail}(\mathsf{const}(a)) = \mathsf{const}(a)$.

We consider another example, for the special case where $A = \mathbb{N}$. We now wish to define (coinductively) the function $\mathsf{from} : \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$ which maps a natural number $n \in \mathbb{N}$ to the sequence $(n, n+1, n+2, n+3, \ldots) \in \mathbb{N}^{\mathbb{N}}$. This involves defining a coalgebra structure $\mathbb{N} \to \mathbb{N} \times \mathbb{N}$ on the domain $\mathbb{N}$ of the function from that we are trying to define. The direct observation that we can make about a "state" $n \in \mathbb{N}$ is n itself, and the next state is then $n + 1$ (in which we can directly observe $n+1$). Repetition then leads to $\mathsf{from}(n)$. Thus we define the function from in the following diagram.

$$\begin{tikzcd}
\mathbb{N} \arrow[r, "\mathsf{from}"] \arrow[d, "\lambda n.\ (n, n+1)"'] & \mathbb{N}^{\mathbb{N}} \arrow[d, "\cong"', "\langle \mathsf{head}, \mathsf{tail}\rangle"] \\
\mathbb{N} \times \mathbb{N} \arrow[r, "id \times \mathsf{from}"'] & \mathbb{N} \times \mathbb{N}^{\mathbb{N}}
\end{tikzcd}$$

It is then determined by the equations $\mathsf{head}(\mathsf{from}(n)) = n$ and $\mathsf{tail}(\mathsf{from}(n)) = \mathsf{from}(n+1)$.

We are now in a position to provide the formal background for the examples of coinductive definitions and proofs in Section 1.3. For instance, the function $\mathsf{merge}\colon A^{\mathbb{N}} \times A^{\mathbb{N}} \to A^{\mathbb{N}}$ which merges two infinite lists into a single one arises as unique function to the final coalgebra $A^{\mathbb{N}}$ in:

$$\begin{tikzcd}
A^{\mathbb{N}} \times A^{\mathbb{N}} \arrow[r, "\mathsf{merge}"] \arrow[d, "\lambda(\alpha,\beta).\ (\mathsf{head}(\alpha), (\beta, \mathsf{tail}(\alpha)))"'] & A^{\mathbb{N}} \arrow[d, "\cong"', "\langle \mathsf{head}, \mathsf{tail}\rangle"] \\
A \times (A^{\mathbb{N}} \times A^{\mathbb{N}}) \arrow[r, "id \times \mathsf{merge}"'] & A \times A^{\mathbb{N}}
\end{tikzcd}$$

Notice that the coalgebra structure on the left that we put on the domain $A^{\mathbb{N}} \times A^{\mathbb{N}}$ of merge corresponds to the defining "coinduction" clauses for merge, as used in Section 1.3. It expresses the direct observation after a merge, together with the next state (about which we make a next direct observation).

It follows from the commutativity of the above diagram that

$$\mathsf{head}(\mathsf{merge}(\alpha, \beta)) = \mathsf{head}(\alpha) \quad \text{and} \quad \mathsf{tail}(\mathsf{merge}(\alpha, \beta)) = \mathsf{merge}(\beta, \mathsf{tail}(\alpha)).$$

The function $\mathsf{even} : A^{\mathbb{N}} \to A^{\mathbb{N}}$ can similarly be defined coinductively, that is, by finality of $A^{\mathbb{N}}$, as follows:

$$\begin{tikzcd}
A^{\mathbb{N}} \arrow[r, "\mathsf{even}"] \arrow[d, "\lambda\alpha.\ (\mathsf{head}(\alpha), \mathsf{tail}(\mathsf{tail}(\alpha)))"'] & A^{\mathbb{N}} \arrow[d, "\cong"', "\langle \mathsf{head}, \mathsf{tail}\rangle"] \\
A \times A^{\mathbb{N}} \arrow[r, "id \times \mathsf{even}"'] & A \times A^{\mathbb{N}}
\end{tikzcd}$$

The coalgebra structure on $A^{\mathbb{N}}$ on the left gives by finality rise to a unique coalgebra homomorphism, called even. By the commutatitivity of the diagram, it satisfies:

$$\mathsf{head}(\mathsf{even}(\alpha)) = \mathsf{head}(\alpha) \qquad \text{and} \qquad \mathsf{tail}(\mathsf{even}(\alpha)) = \mathsf{even}(\mathsf{tail}(\mathsf{tail}(\alpha))).$$

As before, we define

$$\mathsf{odd}(\alpha) = \mathsf{even}(\mathsf{tail}(\alpha)).$$

Next we prove for all α in $A^{\mathbb{N}}$: $\mathsf{merge}(\mathsf{even}(\alpha), \mathsf{odd}(\alpha)) = \alpha$, by showing that $\mathsf{merge} \circ \langle \mathsf{even}, \mathsf{odd} \rangle$ is a homomorphism of coalgebras from $(A^{\mathbb{N}}, \langle \mathsf{head}, \mathsf{tail} \rangle)$ to $(A^{\mathbb{N}}, \langle \mathsf{head}, \mathsf{tail} \rangle)$. The required equality then follows by uniqueness, because the identity function $id \colon A^{\mathbb{N}} \to A^{\mathbb{N}}$ is (trivially) a homomorphism $(A^{\mathbb{N}}, \langle \mathsf{head}, \mathsf{tail} \rangle) \to (A^{\mathbb{N}}, \langle \mathsf{head}, \mathsf{tail} \rangle)$ as well. Thus, all we have to prove is that we have a homomorphism, *i.e.* that

$$\langle \mathsf{head}, \mathsf{tail} \rangle \circ (\mathsf{merge} \circ \langle \mathsf{even}, \mathsf{odd} \rangle) = (id \times (\mathsf{merge} \circ \langle \mathsf{even}, \mathsf{odd} \rangle)) \circ \langle \mathsf{head}, \mathsf{tail} \rangle.$$

This follows from the following two computations.

$$\begin{aligned} \mathsf{head}(\mathsf{merge}(\mathsf{even}(\alpha), \mathsf{odd}(\alpha))) &= \mathsf{head}(\mathsf{even}(\alpha)) \\ &= \mathsf{head}(\alpha). \end{aligned}$$

And:

$$\begin{aligned} \mathsf{tail}(\mathsf{merge}(\mathsf{even}(\alpha), \mathsf{odd}(\alpha))) &= \mathsf{merge}(\mathsf{odd}(\alpha), \mathsf{tail}(\mathsf{even}(\alpha))) \\ &= \mathsf{merge}(\mathsf{odd}(\alpha), \mathsf{even}(\mathsf{tail}(\mathsf{tail}(\alpha)))) \\ &= \mathsf{merge}(\mathsf{even}(\mathsf{tail}(\alpha)), \mathsf{odd}(\mathsf{tail}(\alpha))) \\ &= (\mathsf{merge} \circ \langle \mathsf{even}, \mathsf{odd} \rangle)(\mathsf{tail}(\alpha)). \end{aligned}$$

In Section 1.7, an alternative method for proving facts such as the one above, will be introduced, which is based on the notion of bisimulation.

Clearly, there are formal similarities between algebra maps and coalgebra maps. We leave it to the reader to check that coalgebra maps can be composed as functions, and that the identity function on the carrier of a coalgebra is a map of coalgebras. There is also the following result, which is dual—including its proof—to Lemma 1.5.5.

Lemma 1.6.4 (i) Final coalgebras, if they exist, are uniquely determined (up-to-isomorphism).

(ii) A final coalgebra $Z \to T(Z)$ is a fixed point $Z \xrightarrow{\cong} T(Z)$ of the functor T. □

Final coalgebras are generalizations of greatest fixed points of monotone functions. As for initial algebras, the existence of final coalgebras is more important than their actual (internal) structure. Their use is determined entirely by their finality property, and not by their structure. Often, the existence of a final coalgebra follows from general properties of the relevant functor (and of the underlying category), see *e.g.* [LS81, SP82, Adá03].

The unique existence of a map of coalgebras into a final coalgebra has two aspects: existence, which gives us a principle of definition by coinduction, and uniqueness, which gives us a principle of proof by coinduction. This will be further illustrated in a series of examples, which will occupy the remainder of this section.

Example 1.6.5 It is not hard to show that the final coalgebra of the functor $T(X) = 1 + (A \times X)$ has as carrier the set $A^\infty = A^\star + A^\mathbb{N}$ of finite and infinite lists of A's. The associated "possible next" coalgebra structure

$$\mathsf{pn}\colon A^\infty \longrightarrow 1 + A \times A^\infty \qquad \text{is} \qquad \alpha \mapsto \begin{cases} \kappa(*) & \text{if } \alpha = \langle\rangle \\ \kappa'(a, \alpha') & \text{if } \alpha = a \cdot \alpha' \end{cases}$$

is final: for an arbitrary coalgebra $g\colon U \to 1 + (A \times U)$ of the functor T there is a unique homomorphism of coalgebras $f\colon U \to A^\infty$. Earlier in this section we identified such lists in A^∞ as the observable behaviour for machines whose signature of operations is described by T.

We give some examples of coinductive definitions for such finite and infinite lists. First an easy one, describing an empty list $\mathsf{nil}\colon 1 \to A^\infty$ as the unique coalgebra homomorphisms in the following situation.

$$\begin{tikzcd} 1 \arrow[r, "\mathsf{nil}"] \arrow[d, "\kappa"'] & A^\infty \arrow[d, "\cong"', "\mathsf{pn}"] \\ 1 + (A \times 1) \arrow[r, "id + (id \times \mathsf{nil})"'] & 1 + (A \times A^\infty) \end{tikzcd}$$

This determines nil as $\mathsf{pn}^{-1} \circ \kappa$. We define a prefix operation $\mathsf{cons}\colon A \times A^\infty \to A^\infty$ as $\mathsf{pn}^{-1} \circ \kappa'$.

We can coinductively define a list inclusion function $\mathsf{list\_incl}\colon A^\star \to A^\infty$ via the coalgebra structure $A^\star \to 1 + (A \times A^\star)$ given by

$$\alpha \mapsto \begin{cases} \kappa(*) & \text{if } \alpha = \mathsf{nil} \\ \kappa'(a, \beta) & \text{if } \alpha = \mathsf{cons}(a, \beta) \end{cases}$$

We leave it to the reader to (coinductively) define an infinite list inclusion $A^\mathbb{N} \to A^\infty$.

A next, more serious example, involves the concatenation function $\mathsf{conc}\colon A^\infty \times A^\infty \to A^\infty$ which yields for two lists $x, y \in A^\infty$ a new list $\mathsf{cons}(x, y) \in A^\infty$ which contains the elements of x followed by the elements of y. Coinductively one defines $\mathsf{conc}(x, y)$ by laying down what the possible observations are on this new list $\mathsf{conc}(x, y)$. Concretely, this means that we should define what

$\mathsf{pn}(\mathsf{conc}(x,y))$ is. The intuition we have of concatenation tells us that the possible next $\mathsf{pn}(\mathsf{conc}(x,y))$ is the possible next $\mathsf{pn}(x)$ of x if x is not the empty list (*i.e.* if $\mathsf{pn}(x) \neq \kappa(*) \in 1$), and the possible next $\mathsf{pn}(y)$ of y otherwise. This is captured in the coalgebra structure $\mathsf{conc\_struct}\colon A^\infty \times A^\infty \to 1 + (A \times (A^\infty \times A^\infty))$ given by:

$$(\alpha,\beta) \mapsto \begin{cases} \kappa(*) & \text{if } \mathsf{pn}(\alpha) = \mathsf{pn}(\beta) = \kappa(*) \\ \kappa'(a,(\alpha',\beta)) & \text{if } \mathsf{pn}(\alpha) = \kappa'(a,\alpha') \\ \kappa'(b,(\alpha,\beta')) & \text{if } \mathsf{pn}(\alpha) = \kappa(*) \text{ and } \mathsf{pn}(\beta) = \kappa'(b,\beta'). \end{cases}$$

The concatenation function $\mathsf{conc}\colon A^\infty \times A^\infty \to A^\infty$ that we wished to define arises as unique coalgebra map resulting from $\mathsf{conc\_struct}$.

The interested reader may wish to prove (by uniqueness!) that:

$$\begin{aligned} \mathsf{conc}(x,\mathsf{nil}) = x &= \mathsf{conc}(\mathsf{nil},x) \\ \mathsf{conc}(\mathsf{conc}(x,y),z) &= \mathsf{conc}(x,\mathsf{conc}(y,z)). \end{aligned}$$

This makes A^∞ a monoid. It is clearly not commutative. One can also prove that $\mathsf{conc}(\mathsf{cons}(a,x),y)) = \mathsf{cons}(a,\mathsf{conc}(x,y))$. The easiest way is to show that applying pn on both sides yields the same result. Then we are done, since pn is an isomorphism.

Example 1.6.6 Consider the functor $T(X) = 1 + X$ from Example 1.5.4. Remember that its initial algebra is given by the set $\mathbb{N} = \{0,1,2,\ldots,\}$ of natural numbers with cotuple of zero and successor functions as algebra structure $[0,S]\colon 1+\mathbb{N} \xrightarrow{\cong} \mathbb{N}$.

The final coalgebra $\overline{\mathbb{N}} \xrightarrow{\cong} 1 + \overline{\mathbb{N}}$ of T is the set

$$\overline{\mathbb{N}} = \{0,1,2,\ldots,\} \cup \{\infty\}$$

of natural numbers augmented with an extra element ∞. The final coalgebra structure $\overline{\mathbb{N}} \to 1 + \overline{\mathbb{N}}$ is best called a predecessor pred because it sends

$$0 \mapsto \kappa(*), \qquad n+1 \mapsto \kappa'(n), \qquad \infty \mapsto \kappa'(\infty)$$

where we have written the coprojections κ, κ' explicitly in order to emphasise the $+$-component to which $\mathsf{pred}(x) \in 1 + \overline{\mathbb{N}}$ belongs. This final coalgebra may be obtained by taking as the constant set A a singleton set 1 for the functor $X \mapsto 1 + (A \times X)$ in the previous example. And indeed, the set $1^\infty = 1^\star + 1^{\mathbb{N}}$ is isomorphic to $\overline{\mathbb{N}}$. The "possible next" operations $\mathsf{pn}\colon 1^\infty \to 1 + (1 \times 1^\infty)$ is then indeed the predecessor.

The defining property of this final coalgebra $\mathsf{pred}\colon \overline{\mathbb{N}} \to 1 + \overline{\mathbb{N}}$ says that for every set U with a function $f\colon U \to 1 + U$ there is a unique function $g\colon U \to \overline{\mathbb{N}}$ in the following diagram.

$$\begin{tikzcd} U \arrow[r, dashed, "g"] \arrow[d, "f"'] & \overline{\mathbb{N}} \arrow[d, "\cong"', "\mathsf{pred}"] \\ 1 + U \arrow[r, dashed, "id + g"'] & 1 + \overline{\mathbb{N}} \end{tikzcd}$$

This says that g is the unique function satisfying

$$\mathsf{pred}(g(x)) = \begin{cases} \kappa(*) & \text{if } f(x) = \kappa(*) \\ \kappa'(g(x')) & \text{if } f(x) = \kappa'(x'). \end{cases}$$

This function g gives us the behaviour that one can observe about systems with one button $X \to 1 + X$, as mentioned in the first (coalgebra) example in Section 1.2.

Consider now the function $f\colon \overline{\mathbb{N}} \times \overline{\mathbb{N}} \to 1 + (\overline{\mathbb{N}} \times \overline{\mathbb{N}})$ defined by

$$f(x, y) = \begin{cases} \kappa(*) & \text{if } \mathsf{pred}(x) = \mathsf{pred}(y) = \kappa(*) \\ \kappa'(\langle x', y\rangle) & \text{if } \mathsf{pred}(x) = \kappa'(x') \\ \kappa'(\langle x, y'\rangle) & \text{if } \mathsf{pred}(x) = \kappa(*),\ \mathsf{pred}(y) = \kappa'(y'). \end{cases}$$

This f puts a coalgebra structure on $\overline{\mathbb{N}} \times \overline{\mathbb{N}}$, for the functor $X \mapsto 1 + X$ that we are considering. Hence it gives rise to a unique coalgebra homomorphism $\oplus\colon \overline{\mathbb{N}} \times \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ in the following situation.

$$\begin{tikzcd} \overline{\mathbb{N}} \times \overline{\mathbb{N}} \arrow[r, "\oplus"] \arrow[d, "f"'] & \overline{\mathbb{N}} \arrow[d, "\cong"', "\mathsf{pred}"] \\ 1 + (\overline{\mathbb{N}} \times \overline{\mathbb{N}}) \arrow[r, "id + \oplus"'] & 1 + \overline{\mathbb{N}} \end{tikzcd}$$

Hence $\oplus$ is the unique function $\overline{\mathbb{N}} \times \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ with

$$\mathsf{pred}(x \oplus y) = \begin{cases} \kappa(*) & \text{if } \mathsf{pred}(x) = \kappa(*) = \mathsf{pred}(y) \\ \kappa'(x \oplus y') & \text{if } \mathsf{pred}(x) = \kappa(*), \mathsf{pred}(y) = \kappa'(y') \\ \kappa'(x' \oplus y) & \text{if } \mathsf{pred}(x) = \kappa'(x'). \end{cases}$$

It is not hard to see that $n \oplus m = n + m$ for $n, m \in \mathbb{N}$ and $n \oplus \infty = \infty = \infty \oplus n$, so that $\oplus$ behaves like addition on the "extended" natural numbers in $\overline{\mathbb{N}}$. One easily verifies that this addition function $\oplus\colon \overline{\mathbb{N}} \times \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ is the special case (for $A = 1$) of the concatenation function $\mathsf{conc}\colon A^\infty \times A^\infty \to A^\infty$ that we introduced in the previous example. This special case distinguishes itself in an important aspect: it can be shown that concatenation (or addition) $\oplus\colon \overline{\mathbb{N}} \times \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ on the extended natural numbers is commutative—*e.g.* by uniqueness, or by bisimulation (see [Rut00] for details)—whereas concatenation $\mathsf{conc}\colon A^\infty \times A^\infty \to A^\infty$ in general is not commutative. If A has more than two elements, then $\mathsf{conc}(x,y) \neq \mathsf{conc}(y,x)$, because they give rise to different observations, *e.g.* for both x, y singleton sequence containing different elements.

There exist also coalgebraic treatments of the *real* numbers, see for instance [PP02].

1.7 Proofs by coinduction and bisimulation

In this section, we shall give an alternative formulation for one of the earlier proofs by coinduction. The new proof does not directly exploit (the uniqueness aspect of) finality, but makes use of the notion of *bisimulation*. We also present one new example and then formulate the general case, allowing us to prove equalities on final coalgebras via bisimulations.

We recall from Example 1.6.3 that the final coalgebra of the functor $T(X) = A \times X$ is the set of infinite lists $A^{\mathbb{N}}$ of elements of A with coalgebra structure $\langle \mathsf{head}, \mathsf{tail} \rangle$. A *bisimulation* on this carrier $A^{\mathbb{N}}$ is a relation $\mathcal{R} \subseteq A^{\mathbb{N}} \times A^{\mathbb{N}}$ satisfying

$$(\alpha, \beta) \in \mathcal{R} \Rightarrow \begin{cases} \mathsf{head}(\alpha) = \mathsf{head}(\beta), & \text{and} \\ (\mathsf{tail}(\alpha), \mathsf{tail}(\beta)) \in \mathcal{R} \end{cases}$$

Sometimes we shall also write $\mathcal{R}(\alpha, \beta)$ for $(\alpha, \beta) \in \mathcal{R}$.
Now $A^{\mathbb{N}}$ satisfies the following *coinductive proof principle*, or cpp for short: For all α and β in $A^{\mathbb{N}}$,

$$\text{if } (\alpha, \beta) \in \mathcal{R}, \text{ for some bisimulation } R \text{ on } A^{\mathbb{N}}, \text{ then } \alpha = \beta. \tag{cpp}$$

Before we give a proof of the principle, which will be based on the finality of $A^{\mathbb{N}}$, we illustrate its use by proving, once again, for all α in $A^{\mathbb{N}}$,

$$\mathsf{merge}(\mathsf{even}(\alpha), \mathsf{odd}(\alpha)) = \alpha.$$

To this end, define the following relation on $A^{\mathbb{N}}$:

$$\mathcal{R} = \{(\mathsf{merge}(\mathsf{even}(\alpha), \mathsf{odd}(\alpha)),\ \alpha) \mid \alpha \in A^{\mathbb{N}}\}.$$

In order to prove the above equality it is, by the coinductive proof principle (cpp), sufficient to show that R is a bisimulation. First, for each pair $(\mathsf{merge}(\mathsf{even}(\alpha), \mathsf{odd}(\alpha)), \alpha) \in \mathcal{R}$ we have equal head's:

$$\begin{aligned} \mathsf{head}(\mathsf{merge}(\mathsf{even}(\alpha), \mathsf{odd}(\alpha))) &= \mathsf{head}(\mathsf{even}(\alpha)) \\ &= \mathsf{head}(\alpha). \end{aligned}$$

And secondly, if we have a pair $(\mathsf{merge}(\mathsf{even}(\alpha), \mathsf{odd}(\alpha)), \alpha)$ in $\mathcal{R}$, then applying

tail on both sides yields a new pair in $\mathcal{R}$, since we can rewrite, using that $\mathsf{odd} = \mathsf{even} \circ \mathsf{tail}$,

$$\begin{aligned}\mathsf{tail}(\mathsf{merge}(\mathsf{even}(\alpha), \mathsf{odd}(\alpha))) &= \mathsf{merge}(\mathsf{odd}(\alpha), \mathsf{tail}(\mathsf{even}(\alpha))) \\ &= \mathsf{merge}(\mathsf{even}(\mathsf{tail}(\alpha)), \mathsf{even}(\mathsf{tail}(\mathsf{tail}(\alpha)))) \\ &= \mathsf{merge}(\mathsf{even}(\mathsf{tail}(\alpha)), \mathsf{odd}(\mathsf{tail}(\alpha))).\end{aligned}$$

For a proof of the cpp, let $\mathcal{R}$ be any bisimulation on $A^{\mathbb{N}}$. If we consider $\mathcal{R}$ as a set (of pairs), then it can be supplied with an $A \times (-)$-coalgebra structure by defining a function

$$\gamma: \mathcal{R} \longrightarrow A \times \mathcal{R} \qquad (\alpha, \beta) \mapsto (\mathsf{head}(\alpha), (\mathsf{tail}(\alpha), \mathsf{tail}(\beta))).$$

Note that γ is well-defined since $(\mathsf{tail}(\alpha), \mathsf{tail}(\beta))$ is in $\mathcal{R}$, because $\mathcal{R}$ is a bisimulation. Now it is straightforward to show that the two projection functions

$$\pi_1: \mathcal{R} \longrightarrow A^{\mathbb{N}} \qquad \text{and} \qquad \pi_2: \mathcal{R} \longrightarrow A^{\mathbb{N}}$$

are homomorphisms of coalgebras from $(\mathcal{R}, \gamma)$ to $(A^{\mathbb{N}}, \langle \mathsf{head}, \mathsf{tail} \rangle)$. Therefore it follows from the finality (*cf.* Definition 1.6.2) of $A^{\mathbb{N}}$ that $\pi_1 = \pi_2$. That is, if $(\alpha, \beta) \in \mathcal{R}$ then $\alpha = \beta$.

The above definition of a bisimulation is a special instance of the following categorical definition of bisimulation, which was introduced by [AM89], and which applies to coalgebras of arbitrary functors T.

Definition 1.7.1 Let T be a functor and let $(X,\, \alpha_X: X \to T(X))$ and $(Y,\, \alpha_Y: Y \to T(Y))$ be two T-coalgebras. A T-*bisimulation* between $(X,\, \alpha_X)$ and $(Y,\, \alpha_Y)$ is a relation $\mathcal{R} \subseteq X \times Y$ for which there exists a T-coalgebra structure $\gamma: \mathcal{R} \to T(\mathcal{R})$ such that the two projection functions $\pi_1: \mathcal{R} \to X$ and $\pi_2: \mathcal{R} \to Y$ are homomorphisms of T-coalgebras:

$$\begin{tikzcd} X \arrow[d, "\alpha_X"'] & \mathcal{R} \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma"] & Y \arrow[d, "\alpha_Y"] \\ T(X) & T(\mathcal{R}) \arrow[l, "T(\pi_1)"] \arrow[r, "T(\pi_2)"'] & T(Y) \end{tikzcd}$$

We call a bisimulation between a coalgebra $(X,\, \alpha_X)$ and itself a bisimulation *on* X. And we use the following notation:

$$x \sim x' \iff \text{there exists a } T\text{-bisimulation } \mathcal{R} \text{ on } X \text{ with } (x, x') \in \mathcal{R}$$

The general formulation of the coinduction proof principle is now as follows.

Theorem 1.7.2 Let $c: Z \xrightarrow{\cong} T(Z)$ be the final T-coalgebra. For all z and z' in Z,

$$\text{if } z \sim z' \text{ then } z = z'. \qquad \textit{(cpp)}$$

As in the example above, the proof of this principle is immediate by finality: both the projections π_1 and π_2 are homomorphisms from $(\mathcal{R}, \gamma)$ to the final coalgebra (Z, c). By finality, $\pi_1 = \pi_2$, which proves the theorem.

This general version of the coinduction proof principle is surprisingly powerful, notwithstanding the fact that the proof of cpp is almost trivial. The reader is referred to [Rut00] for further examples of definitions and proofs by coinduction. In Section 1.8, we shall see how this coalgebraic notion of bisimulation coincides with the classical notion of Park and Milner for the case of processes.

There exist other formalisations of the notion of bisimulation: in [HJ98] a bisimulation is described as a coalgebra in a category of relations, for a suitably lifted functor (associated with the original functor T); in the context of (coalgebraic) modal logic the notion of behavioural equivalence if often used, see *e.g.* [CP07, Kli07]; in [JNW96], bisimulations occur as spans of so-called open maps; and in [Bar03, CHL03], stronger versions of bisimulation (and coinduction) are given called λ-bisimulations. But in a set-theoretic context, the above definition seems to be most convenient. Simulations, or “bisimulations” in one direction only, are described in [HJ04].

The above categorical definition of bisimulation can be seen to be the formal (categorical) dual of the notion of *congruence* on algebras, which for T-algebras (U, a) and (V, b) can be defined as a relation $\mathcal{R} \subseteq U \times V$ for which there exists a T-algebra structure $c: T(\mathcal{R}) \to \mathcal{R}$ such that the two projection functions $\pi_1: \mathcal{R} \to U$ and $\pi_2: \mathcal{R} \to V$ are homomorphisms of T-algebras:

$$\begin{tikzcd}
T(U) \arrow[d, "a"'] & T(\mathcal{R}) \arrow[l, "T(\pi_1)"'] \arrow[r, "T(\pi_2)"] \arrow[d, "c"'] & T(V) \arrow[d, "b"] \\
U & \mathcal{R} \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & V
\end{tikzcd}$$

Using the above notions of congruence on algebras and bisimulation on coalgebras, the duality between induction and coinduction can be succinctly expressed as follows. For *initial algebras* (A, a), we have:

$$\text{for every congruence relation } \mathcal{R} \subseteq A \times A\,, \quad \Delta_A \subseteq \mathcal{R}$$

where $\Delta_A = \{(a, a) \mid a \in A\}$ is the diagonal on A. Dually, for *final coalgebras* (Z, z) we have the following:

$$\text{for every bisimulation relation } \mathcal{R} \subseteq Z \times Z\,, \quad \mathcal{R} \subseteq \Delta_Z$$

One can show that the above property of initial algebras is precisely the familiar induction principle on algebras such as the natural numbers. (The above property of final coalgebras is trivially equivalent to the formulation of Theorem 1.7.2).) We refer the reader to [Rut00, Section 13] for further details.

1.8 Processes coalgebraically

In this section, we shall present labelled transition systems as coalgebras of a certain "behaviour" functor B. We shall see that the corresponding coalgebraic notion of bisimulation coincides with the classical notion of Park and Milner. Finally, we shall introduce the final coalgebra for the functor B, the elements of which can be seen as (canonical representatives of) *processes*.

A (possibly nondeterministic) transition system $(X, A, \longrightarrow_X)$ consists of a set X of states, a set A of transition labels, and a transition relation $\longrightarrow_X \subseteq X \times A \times X$. As usual, we write $x \xrightarrow{a}_X x'$ for transitions $(x, a, x') \in \longrightarrow_X$.

Consider the functor B defined by

$$B(X) = \mathcal{P}(A \times X) = \{V \mid V \subseteq A \times X\}$$

A labeled transition system $(X, A, \longrightarrow_X)$ can be identified with a B-coalgebra $(X, \alpha_X\colon X \to B(X))$, by putting

$$(a, x') \in \alpha_X(x) \quad \Longleftrightarrow \quad x \xrightarrow{a}_X x'$$

In other words, the class of all labeled transition systems coincides with the class of all B-coalgebras. Let $(X, A, \longrightarrow_X)$ and $(Y, A, \longrightarrow_Y)$ be two labeled transition systems with the same set A of labels. An interesting question is what a coalgebra homomorphism between these two transition systems (as coalgebras (X, α_X) and (Y, α_Y)) is, in terms of the transition structures $\longrightarrow_X$ and $\longrightarrow_Y$. Per definition, a B-homomorphism $f\colon (X, \alpha_X) \to (Y, \alpha_Y)$ is a function $f\colon X \to Y$ such that $B(f) \circ \alpha_X = \alpha_Y \circ f$, where the function $B(f)$, also denoted by $\mathcal{P}(A \times f)$, is defined by

$$B(f)(V) = \mathcal{P}(A \times f)(V) = \{\langle a, f(s)\rangle \mid \langle a, s\rangle \in V\}.$$

One can easily prove that the equality $B(f) \circ \alpha_X = \alpha_Y \circ f$ is equivalent to the following two conditions, for all $x \in X$:

(1) for all x' in X, if $x \xrightarrow{a}_X x'$ then $f(x) \xrightarrow{a}_Y f(x')$;
(2) for all y in Y, if $f(x) \xrightarrow{a}_Y y$ then there is an x' in X with $x \xrightarrow{a}_X x'$ and $f(x') = y$.

Thus a homomorphism is a function that preserves and reflects transitions.

This notion is quite standard, but sometimes only preservation is required, see *e.g.* [JNW96].

There is the following well-known notion of bisimulation for transition systems [Mil89, Par81]: a bisimulation between between transition systems X and Y (as above) is a relation $\mathcal{R} \subseteq X \times Y$ satisfying, for all $(x, y) \in \mathcal{R}$,

(i) for all x' in X, if $x \xrightarrow{a}_X x'$ then there is y' in Y with $y \xrightarrow{a}_Y y'$ and $(x', y') \in \mathcal{R}$;
(ii) for all y' in Y, if $y \xrightarrow{a}_Y y'$ then there is x' in X with $x \xrightarrow{a}_X x'$ and $(x', y') \in \mathcal{R}$.

For the relation between this notion of bisimulation and the notion of zig-zag relation from modal logic, see Chapter SANGIORGI.
The coalgebraic notion of B-bisimulation (Definition 1.7.1) coincides with the above definition: If $\mathcal{R}$ is a B-bisimulation then conditions (i) and (ii) follow from the fact that both π_1 and π_2 are homomorphisms. Conversely, any relation $\mathcal{R}$ satisfying (i) and (ii) above can be seen to be a B-bisimulation by defining a coalgebra structure $\gamma : \mathcal{R} \to B(\mathcal{R})$ as follows:

$$\gamma(x, y) = \; \{ < a, (x', y') > \mid \; x \xrightarrow{a}_X x' \;\text{ and }\; y \xrightarrow{a}_Y y' \;\text{ and }\; (x', y') \in \mathcal{R} \; \}$$

One then readily proves that π_1 and π_2 are homomorphisms.

A concrete example of a bisimulation relation between two transition systems X and Y is the following. Consider two systems X and Y:

$$X = \left(\begin{tikzcd} x_0 \arrow[r, "b"] \arrow[d, "a"'] & x_1 \arrow[r, "b"] \arrow[d, "a"'] & \cdots \\ x_0' & x_1' & \end{tikzcd} \right) \qquad Y = \left(\begin{tikzcd} y \arrow[loop right, "b"] \arrow[d, "a"'] \\ y' \end{tikzcd} \right)$$

The relation

$$\{ \; (x_i, x_j) \mid i, j \geq 0\} \; \cup \; \{(x_i', x_j') \mid i, j \geq 0\}$$

is then a bisimulation on X. And

$$\mathcal{R} \; = \; \{(x_i, y) \mid i \geq 0\} \; \cup \; \{(x_i', y') \mid i \geq 0\}$$

is a bisimulation between X and Y. The latter relation $\mathcal{R}$ is called a *functional* bisimulation because it is the graph

$$\{(x, f(x)) \mid x \in X \}$$

of a homomorphism $f \colon X \to Y$ defined by $f(x_i) = y$ and $f(x_i') = y'$. Note that there exists no homomorphism in the reverse direction from Y to X.

For cardinality reasons, a final B-coalgebra cannot exist: by Lemma 1.6.4 (ii), any final coalgebra is a fixed point: $X \cong \mathcal{P}(A \times X)$, and such a set does not exist because the cardinality of $\mathcal{P}(A \times X)$ is strictly greater than that of X (for non-empty sets of labels A). Therefore we restrict to so-called *finitely branching* transition systems, satisfying, for all states s,

$$\{\langle a, s\rangle \mid s \xrightarrow{a}_X s'\} \text{ is finite.}$$

Such systems can be identified with coalgebras of the functor

$$B_f(X) = \mathcal{P}_f(A \times X) = \{V \subseteq A \times X \mid V \text{ is finite}\}.$$

For this functor, a final coalgebra *does* exist. The proof, which is a bit technical, is due to Barr [Bar93] (see also [RT94, Rut00]), and is omitted here (*cf.* the discussion in Section 1.2).

In what follows, let (Π, π) be the final B_f-coalgebra, which is unique up to isomorphism. Borrowing the terminology of concurrency theory, we call the elements of Π *processes* and denote them by P, Q, R. As before, we shall denote transitions by

$$P \xrightarrow{a} Q \iff (a, Q) \in \pi(P)$$

Being a final coalgebra, (Π, π) satisfies the coinduction proof principle (Theorem 1.7.2): for all $P, Q \in \Pi$,

$$\text{if } P \sim Q \text{ then } P = Q.$$

The following theorem shows that we can view the elements of Π as canonical, minimal representatives of (finitely branching) labeled transition systems.

Theorem 1.8.1 Let (X, α_X) be a B_f-coalgebra, that is, a finitely branching labeled transition system. By finality, there is a unique homomorphism $f : (X, \alpha_X) \to (\Pi, \pi)$. It satisfies, for all $x, x' \in X$:

$$x \sim x' \iff f(x) = f(x')$$

Proof The implication from left to right follows from the fact that homomorphisms are (functional) bisimulations and the coinduction proof principle. For the implication from right to left, note that

$$\mathcal{R} = \{(x, x') \in X \times X \mid f(x) = f(x')\}$$

is a bisimulation relation on X. □

In conclusion of the present section, we define a number of operators on processes by coinduction, and then prove various of their properties by the coinduction proof principle.

As a first example, we define a non-deterministic merge operation on processes. To this end, we supply $\Pi \times \Pi$ with a B_f-coalgebra structure

$$\mu : \Pi \times \Pi \to B_f(\Pi \times \Pi)$$

defined by

$$\mu\langle P, Q\rangle \;=\; \{\langle a, \langle P', Q\rangle\rangle \mid P \xrightarrow{a} P'\} \cup \{\langle a, \langle P, Q'\rangle\rangle \mid \; Q \xrightarrow{a} Q'\}. \tag{1.5}$$

By finality of Π, there exists a unique B_f-homomorphism

$$\mathsf{merge}\colon \Pi \times \Pi \to \Pi.$$

We shall use the following standard notation:

$$P \mid Q \equiv \mathsf{merge}\langle P, Q\rangle$$

It follows from the fact that merge is a homomorphism of transition systems, *i.e.*, from

$$B_f(\mathsf{merge}) \circ \mu = \pi \circ \mathsf{merge}$$

that it satisfies precisely the following rules:

$$\frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q} \qquad \text{and} \qquad \frac{Q \xrightarrow{a} Q'}{P \mid Q \xrightarrow{a} P \mid Q'} \tag{1.6}$$

The function $\mid$ satisfies a number of familiar properties. Let $\mathbf{0}$ be the *terminated* process: formally,

$$\mathbf{0} = \pi^{-1}(\emptyset)$$

for which no transitions exist. The following equalities

(1) $\mathbf{0} \mid P = P$;
(2) $P \mid Q = Q \mid P$;
(3) $(P \mid Q) \mid R = P \mid (Q \mid R)$,

are a consequence of (cpp) and the fact that the following relations are bisimulations on Π:

(a) $\{(\mathbf{0} \mid P,\, P) \mid P \in \Pi\}$;
(b) $\{(P \mid Q,\, Q \mid P) \mid P, Q \in \Pi\}$;
(c) $\{((P \mid Q) \mid R,\, P \mid (Q \mid R)) \mid P, Q, R \in \Pi\}$.

For instance, the first relation (a) is a bisimulation because we have transitions, for any P in Π:

$$\mathbf{0} \mid P \xrightarrow{a} \mathbf{0} \mid P' \qquad \text{if and only if} \qquad P \xrightarrow{a} P',$$

and $(\mathbf{0} \mid P',\, P')$ is again in the relation. For the second relation (b), consider a pair of processes $(P \mid Q,\, Q \mid P)$, and suppose that we have a transition step

$$P \mid Q \xrightarrow{a} R,$$

for some process R in Π. (The other case, where a first step of $Q \mid P$ is considered, is proved in exactly the same way.) It follows from the definition of merge that one of the following two situations applies: either there exists a transition $P \xrightarrow{a} P'$ and $R = P' \mid Q$, or there exists a transition $Q \xrightarrow{a} Q'$ and $R = P \mid Q'$. Let us consider the first situation, the second being similar. If $P \xrightarrow{a} P'$ then it follows again from the rules above that there exists also a transition

$$Q \mid P \xrightarrow{a} Q \mid P'.$$

But then we have mimicked the transition step of $P \mid Q$ by a transition step of $Q \mid P$, in such a way that the resulting processes are again in the relation:

$$(P' \mid Q,\, Q \mid P')$$

is again a pair in relation (b). This shows that also relation (b) is a bisimulation. For (c), the same kind of argument can be given.

Let us return for a moment to the coinductive definition of the merge operator above. There is a very close correspondence between the two transition rules (1.6) and the definition (1.5) of the coalgebra structure μ on $\Pi \times \Pi$. In fact, we could take the transition rules as a specification of the merge operator we were after; then use these rules to define μ as above; and finally define the merge operator by the homomorphism into Π, as we did above.

We illustrate this approach by the coinductive definition of a number of process operators at the same time, which together constitute a simple CCS-like process calculus. Let the set Exp of syntactic expressions (denoted by E, F etc.) be given by

$$\begin{array}{rcll} E & ::= & \hat{0} & \\ & \mid & \hat{a} & (\text{for every } a \in A) \\ & \mid & \hat{P} & (\text{for every } P \in \Pi) \\ & \mid & E \mathbin{\hat{|}} F & \\ & \mid & E \mathbin{\hat{+}} F & \\ & \mid & E \mathbin{\hat{;}} F & \end{array}$$

Here we use the symbol *hat* to indicate that we are dealing with syntactic entities. For instance, for every process $P \in \Pi$, the set Exp contains a syntactic expression $\hat{P}$. Thus we have mappings $A \to Exp$ and $\Pi \to Exp$.

Next we define a transition relation on Exp by the following axioms and rules:

$$\hat{a} \xrightarrow{a} \hat{0}$$

$$\hat{P} \xrightarrow{a} \hat{Q} \iff P \xrightarrow{a} Q \qquad (\iff (a, Q) \in \pi(P))$$

$$\frac{E \xrightarrow{a} E'}{E \mathbin{\hat{+}} F \xrightarrow{a} E'} \qquad \frac{F \xrightarrow{a} F'}{E \mathbin{\hat{+}} F \xrightarrow{a} F'}$$

$$\frac{E \xrightarrow{a} E'}{E \mathbin{\hat{|}} F \xrightarrow{a} E' \mathbin{\hat{|}} F} \qquad \frac{F \xrightarrow{a} F'}{E \mathbin{\hat{|}} F \xrightarrow{a} E \mathbin{\hat{|}} F'}$$

$$\frac{E \xrightarrow{a} E'}{E \mathbin{\hat{;}} F \xrightarrow{a} E' \mathbin{\hat{;}} F} \qquad \frac{E \not\rightarrow \text{ and } F \xrightarrow{a} F'}{E \mathbin{\hat{;}} F \xrightarrow{a} F'}$$

Having such a transition structure on Exp, we can define a B_f-coalgebra structure $\gamma : Exp \to B_f(Exp)$ by

$$\gamma(E) \;=\; \{\langle a, F\rangle \mid E \xrightarrow{a} F\}$$

Note that by construction $\hat{\ } : \Pi \to Exp$ is now a coalgebra homomorphism.

By finality of Π, there exists a unique homomorphism $h : (Exp, \gamma) \to (\Pi, \pi)$ which assigns to each syntactic expression E a corresponding process $h(E) \in \Pi$. We can use it to define semantic operators on Π corresponding to the syntactic operators on Exp, as follows:

$$\begin{array}{lcr}
\mathbf{0} & \stackrel{\text{def}}{=} & h(\hat{0}) \\
\mathbf{a} & \stackrel{\text{def}}{=} & h(\hat{a}) \\
P \mid Q & \stackrel{\text{def}}{=} & h(\hat{P} \mathbin{\hat{|}} \hat{Q}) \\
P + Q & \stackrel{\text{def}}{=} & h(\hat{P} \mathbin{\hat{+}} \hat{Q}) \\
P; Q & \stackrel{\text{def}}{=} & h(\hat{P} \mathbin{\hat{;}} \hat{Q})
\end{array} \tag{1.7}$$

In this manner, we have obtained three operators on processes P and Q: the merge $P \mid Q$; the choice $P + Q$; and the sequential composition $P; Q$. (It is straightforward to check that the present definition of the merge coincides with the one given earlier.) The constant $\mathbf{0}$ is defined as the process that cannot make any transitions (since the transition relation on Exp does not specify any transitions for $\hat{0}$). As a consequence, $\mathbf{0}$ coincides with the terminated process (also denoted by $\mathbf{0}$) introduced earlier. The constant $\mathbf{a}$ denotes a process that can take a single a-step to $\mathbf{0}$ and then terminates. Furthermore it is worth noticing that the homomorphism h acts as the identity on processes; that is, $h(\hat{P}) = P$, which can be easily proved by (ccp) or directly by finality of Π. Also note that it is possible to add recursive process definitions to the above, see for instance [RT94]. This would allow us to use guarded equations such as $X = \hat{a} \,\hat{;}\, X \mathbin{\hat{+}} \hat{b}$, which would define a process $P = h(X)$ with transitions $P \xrightarrow{a} P$ and $P \xrightarrow{b} \mathbf{0}$.

In the above, we have exploited the finality of the set Π of all processes to define constants and operators by coinduction. Essentially the same procedure can be followed to define operators on various other structures such as, for instance, formal languages and power series [Rut03] and binary trees [SR07].

A question that naturally arises is under which conditions the above type of definition scheme works, that is, when does it uniquely determine the operators one wants to define. As it turns out, this very much depends on the syntactic shape of the defining equations or, in terms of the transition relation defined on Exp above, on the shape of the axioms and rules used to specify the transition relation. There is in fact a close relationship between the various syntactic transition system specification *formats* studied in the literature (such as GSOS, tyft-tyxt, and the like), on the one hand, and well-formed coinductive definition schemes, on the other hand.

In conclusion of this section, let us indicate how the above construction of semantic operators out of operational specifications can be put into a general categorical perspective. First of all, we observe that the set Exp of expressions is the initial algebra of the functor

$$T(X) = 1 + A + \Pi + (X \times X) + (X \times X) + (X \times X)$$

where the constituents on the right correspond, respectively, to the constant symbol $\hat{0}$; the elements $\hat{a}$ with $a \in A$; the elements $\hat{P}$ with $P \in \Pi$; and the three operations of merge, choice, and sequential composition. Above we had already supplied the set Exp with a B_f-coalgebra structure (Exp, γ). Therefore Exp is a so-called *bialgebra*: a set which has both an algebra and a coalgebra structure. Similarly, the definition of the semantic operators above (1.7) supplies Π, which was defined as the final B_f-coalgebra, with a T-algebra structure, turning it thereby into a bialgebra as well. The relationship between the T-algebra and B_f-coalgebra structures on Exp and Π is provided by the fact that the mapping h above is *both* a homomorphism of B_f-coalgebras *and* a homomorphism of

T-algebras. All of which can be pleasantly expressed by the following diagram:

$$\begin{tikzcd}
T(Exp) \arrow[r, "T(h)"] \arrow[d, "\cong"'] & T(\Pi) \arrow[d] \\
Exp \arrow[r, "h"] \arrow[d] & \Pi \arrow[d, "\cong"] \\
B_f(Exp) \arrow[r, "B_f(h)"'] & B_f(Y)
\end{tikzcd}$$

The interplay between algebra (syntactic operators) and coalgebra (their behaviour) has become an important topic of research within the coalgebra community. For further reading see for instance [RT94] and [TP97, Bar03, Jac06]. In these latter references, natural transformations called *distributive laws* are used to relate the syntax functor and the behaviour functor. Compositionality then comes for free. Other examples of the interplay between algebraic and coalgebraic structure include recent generalisations of Kleene's theorem and Kleene algebra to large families of coalgebras, including processes, Mealy machines, and weighted and probabilistic systems [BRS09, BBRS09].

1.9 Trace Semantics, coalgebraically

Let $\longrightarrow \subseteq X \times A \times X$ be a transition system as in the previous section. As we have seen, it may be written as a coalgebra $(X, \alpha\colon X \to \mathcal{P}(A \times X))$. An execution is a sequence of consecutive transition steps:

$$x_0 \xrightarrow{a_0} x_1 \xrightarrow{a_1} x_2 \xrightarrow{a_2} \cdots$$

A trace of this transition system is then a sequence $\langle a_0, a_1, a_2, \ldots \rangle$ of actions occurring in such an execution.

This section describes a systematic way to capture such traces coalgebraically, following [HJ05, HJS07], which is general and generic. Here we concentrate on the powerset case. This involves both initial algebras and final coalgebras, in different "universes". The description involves a number of preparatory steps.

Finite executions and traces

Our description applies to finite traces. In order to capture them we introduce an additional symbol X for succesful termination. We can do so by considering coalgebras of a slightly different functor, namely $\mathcal{P}(1 + A \times X)$ with additional singleton set $1 = \{\checkmark\}$. For a coalgebra $\alpha\colon X \to \mathcal{P}(1 + A \times X)$ we then write $x \xrightarrow{a} x'$ if $(a, x') \in \alpha(x)$ and $x \to \checkmark$ if $\checkmark \in \alpha(x)$.

We shall write $F(X) = 1 + A \times X$ for the functor inside the powerset. Hence we concentrate on $\mathcal{P}F$-coalgebras.

A finite execution is one that ends with $\checkmark$, as in $x_0 \xrightarrow{a_0} x_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} x_n \to \checkmark$. A trace is called finite if it comes from such a finite execution. We shall write $\mathsf{tr}(x)$ for the set of finite traces of executions that start in $x \in X$. This yields a function $\mathsf{tr}\colon X \to \mathcal{P}(A^\star)$ where $A^\star$ is the set of finite sequences of elements of A. Notice that $A^\star$ is the initial algebra of the functor F, see Example 1.5.6.

A category of relations

A map of the form $X \to \mathcal{P}(Y)$ can be seen as a "non-deterministic" function that yields a set of outcomes in Y for a single input element from X. It may be identified with a relation between X and Y, *i.e.* with a subset of the product $X \times Y$. We can understand such a function/relation as an arrow in a category *Rel* of sets and relations.

This category *Rel* has ordinary sets as objects. Morphism $X \to Y$ in *Rel* are ordinary functions $X \to \mathcal{P}(Y)$. They can be composed via "relational" composition. For arrows $f\colon X \to Y$ and $g\colon Y \to Z$ in *Rel* we have to form $g \circ f\colon X \to Z$. As ordinary functions we define $g \circ f\colon X \to \mathcal{P}(Z)$ for $f\colon X \to \mathcal{P}(Y)$ and $g\colon Y \to \mathcal{P}(Z)$, by:

$$(g \circ f)(x) \quad = \quad \bigcup\{g(y) \mid y \in f(x)\}. \tag{1.8}$$

Notice that our notation is now (deliberately) ambigous, depending on the category (or universe) in which we work. An arrow $X \to Y$ in *Set* is an ordinary function from X to Y, but an arrow $X \to Y$ in *Rel* is a "non-deterministic" function $X \to \mathcal{P}(Y)$.

It is not hard to see that the singleton map $\{-\}\colon X \to \mathcal{P}(X)$ forms the identity map $X \to X$ on the object X in the category *Rel*. Explicitly, $\{-\} \circ f = f$ and $g \circ \{-\} = g$, for relational composition $\circ$.

There is an obvious functor $J\colon \mathit{Set} \to \mathit{Rel}$ that sends a set X to itself and a function $f\colon X \to Y$ to its graph relation $\{(x, f(x)) \mid x \in X\}$. This graph may be identified with the composite $X \xrightarrow{f} Y \xrightarrow{\{-\}} \mathcal{P}(Y)$.

Lifting the functor

For the main result of this section we need to consider a $\mathcal{P}F$-coalgebra $(X, \alpha\colon X \to \mathcal{P}(F(X)))$ as an F-coalgebra $(X, \alpha\colon X \to F(X))$, by moving to the category

Rel. Indeed, a map $X \to \mathcal{P}(F(X))$ is an arrow $X \to F(X)$ in *Rel*. In order to understand it as coalgebra we need to know that F is—or lifts to—a functor $\overline{F}: Rel \to Rel$. It is obvious how such lifting works on objects, namely $\overline{F}(X) = F(X)$, but how it works on morphisms in *Rel* requires some care.

For an arrow $X \to Y$ in *Rel* we need to construct a new map $FX \to FY$ in *Rel*. That is, a function $f: X \to \mathcal{P}(Y)$ yields a function $\overline{F}(f): FX \to \mathcal{P}(F(Y))$, defined as follows.

$$\begin{array}{rcl} 1 + A \times X & \xrightarrow{\overline{F}(f)} & \mathcal{P}(1 + A \times Y) \\ \mathsf{X} & \longmapsto & \{\mathsf{X}\} \\ (a, x) & \longmapsto & \{(a, y) \mid y \in f(x)\}. \end{array}$$

It is not hard to check that this "lifting" of $F: Set \to Set$ to $\overline{F}: Rel \to Rel$ indeed preserves identities and composition—from *Rel*. It yields a commuting diagram:

$$\begin{tikzcd} Rel \arrow[r, "\overline{F}"] & Rel \\ Set \arrow[u, "J"] \arrow[r, "F"] & Set \arrow[u, "J"'] \end{tikzcd} \tag{1.9}$$

The trace theorem

The following result (from [HJ05]) combines initial algebras and final coalgebras for a description of trace semantics.

Theorem 1.9.1 The initial F-algebra in *Set* yields a final $\overline{F}$-coalgebra in *Rel*. □

There are several ways to prove this result. A particular snappy proof is presented in [HJS07, Theorem 3.8]; it uses adjunctions and is therefore outside the scope of this paper. Here we omit the proof and concentrate on the relevance of this result for trace semantics.

So we first spell out what is actually stated in the theorem. Recall from Example 1.5.6 that the initial algebra of the functor $F = 1 + A \times (-)$ is the set $A^\star$ of finite sequences of elements in A with operations $[\mathsf{nil}, \mathsf{cons}]: 1 + A \times A^\star \xrightarrow{\cong} A^\star$. Since this initial algebra is an isomorphism, we can consider its inverse $A^\star \xrightarrow{\cong} F(A^\star)$, formally as an isomorphism in the category *Set*. By applying the functor $J: Set \to Rel$ from the diagram (1.9) this yields an isomorphism in *Rel*

$$A^\star = J(A^\star) \xrightarrow{\cong} J(F(A^\star)) = \overline{F}(J(A^\star)) = \overline{F}(A^\star).$$

which is an $\overline{F}$-coalgebra (recall that $\overline{F}$ is F and J the identity on objects/sets). The theorem claims that this map is the final $\overline{F}$-coalgebra in *Rel*.

In order to clarify this switch of categories we go a bit deeper into the details. Let's write $\beta = [\mathsf{nil}, \mathsf{cons}]^{-1}: A^\star \xrightarrow{\cong} 1 + A \times A^\star = F(A^\star)$ for the inverse of the initial algebra (in *Set*). It yields $J(\beta) = \{-\} \circ \beta: A^\star \to \mathcal{P}(F(A^\star))$. We claim that this β is an isomorphism $A^\star \to \overline{F}(A^\star)$, an isomorphism *in the category Rel!* This $J(\beta)$ may not look like an isomorphism, but we have to keep in mind that composition in *Rel*, as described in (1.8), is different from composition in *Set*. In general, for an isomorphism $f: Y \to X$ in *Set* have in *Rel*:

$$\begin{array}{rcll}\Big(J(f) \circ J(f^{-1})\Big)(x) & = & \bigcup\Big\{J(f)(y) \mid y \in J(f^{-1})(x)\Big\} & \text{by (1.8)} \\ & = & \bigcup\Big\{\{f(y)\} \mid y \in \{f^{-1}(x)\Big\} & \\ & = & \{f(f^{-1}(x))\} & \\ & = & \{x\} & \\ & = & id_X(x) & \end{array}$$

where the latter identity map id_X is the identity in *Rel*, given by the singleton map.

Assume now that we have a transition system $\alpha: X \to \mathcal{P}(1 + A \times X)$ as in the beginning of this section. We may now understand it as an $\overline{F}$-coalgebra $X \to \overline{F}(X)$. We shall do so and continue to work in the category *Rel*. The finality claimed in the theorem then yields a unique homomorphism in:

$$\begin{tikzcd} X \arrow[r, "\mathsf{tr}"] \arrow[d, "\alpha"'] & A^\star \arrow[d, "\cong"] \\ \overline{F}(X) \arrow[r, "\overline{F}(\mathsf{tr})"] & \overline{F}(A^\star) \end{tikzcd}$$

It is essential that this is a diagram in the category *Rel*. We shall unravel what commutation means, using composition as in (1.8). The composite $\overline{F}(\mathsf{tr}) \circ \alpha$ is the relation on $X \times (1 + A \times A^\star)$ consisting of:

$$\{(x, \mathsf{X}) \mid \mathsf{X} \in \alpha(x)\} \;\cup\; \{(x, (a, \sigma)) \mid \exists x'.\ (a, x') \in \alpha(x) \wedge \sigma \in \mathsf{tr}(x')\}.$$

The other composition, $\cong \circ\, \mathsf{tr}$, yields the relation:

$$\{(x, \mathsf{X}) \mid \mathsf{nil} \in \mathsf{tr}(x)\} \;\cup\; \{(x, (a, \sigma)) \mid \mathsf{cons}(a, \sigma) \in \mathsf{tr}(x)\}.$$

The equality of these sets yields the defining clauses for trace semantics, namely:

$$\begin{array}{rcl}\mathsf{nil} \in \mathsf{tr}(x) & \Longleftrightarrow & x \to \mathsf{X} \\ \mathsf{cons}(a, \sigma) \in \mathsf{tr}(x) & \Longleftrightarrow & \exists x'.\ x \xrightarrow{a} x' \wedge \sigma \in \mathsf{tr}(x').\end{array}$$

where we have used the transition notation for the coalgebra α.

What we thus see is that the abstract idea of finality (for coalgebras) makes sense not only in the standard category/universe of sets and functions, but also in the world of sets and relations. This genericity can only be formulated and appreciated via the language of categories. It demonstrates clearly why the theory of coalgebras relies so heavily on category theory.

Theorem 1.9.1 allows for considerable generalisation. The crucial aspect of the powerset that is used is that it is a so-called monad, with the category *Rel* as its "Kleisli" category. The result may be formulated more generally for suitable monads and functors F, see [HJS07]. It can then also be applied to probabilistic transition systems, and even to a combination of probabilistic and possibilistic (non-deterministic) systems (see [Jac08]).

1.10 Exercises

Exercise 1.10.1 Use initiality to define a function $sum : \mathbb{N}^\star \to \mathbb{N}$ such that $sum(\mathsf{nil}) = 0$ and

$$sum(a_1, \ldots, a_n) = a_1 + \cdots + a_n$$

for all $(a_1, \ldots, a_n) \in \mathbb{N}^\star$.

Exercise 1.10.2 Use initiality to define a function $L : \mathbb{N} \to \mathbb{N}^\star$ such that

$$L(n) = (0, 1, 2, \ldots, n)$$

for all $n \geq 0$. Same question, now for $H : \mathbb{N} \to \mathbb{N}^\star$ such that $H(0) = \langle\rangle$ and

$$H(n) = (0, 1, 2, \ldots, n - 1)$$

for all $n \geq 1$.

Exercise 1.10.3 A preorder $(P, \leq)$ can be viewed as a category. Objects are the elements of P and we have an arrow $p \to q$ iff $p \leq q$. An order-preserving *function* $f : P \to P$ can then be seen as a *functor* from the category P to itself. Show that least fixed points of f are precisely the initial algebras and that greatest fixed points correspond to final coalgebras.

Exercise 1.10.4 The set of (finite) binary trees with (node) labels in a given set A can be defined as the initial algebra BT_A of the functor

$$B(X) = 1 + (X \times A \times X)$$

Use initiality to define a function $size : BT_A \to \mathbb{N}$. Next use initiality to define two tree traversal functions of type $BT_A \to A^\star$ that flatten a tree into a word consisting of its labels: one depth-first and one breadth-first.

Exercise 1.10.5 Let us call a relation $\mathcal{R} \subseteq \mathbb{N} \times \mathbb{N}$ a *congruence* if $(0,0) \in \mathcal{R}$ and

$$\text{if} \quad (n,m) \in \mathcal{R} \quad \text{then} \quad (S(n), S(m)) \in \mathcal{R}$$

for all $(n,m) \in \mathbb{N} \times \mathbb{N}$.

(i) Show that any relation $\mathcal{R} \subseteq \mathbb{N} \times \mathbb{N}$ is a congruence iff there exists an algebra structure $\rho : 1 + \mathcal{R} \to \mathcal{R}$ such that the projections $\pi_1 : \mathcal{R} \to \mathbb{N}$ and $\pi_2 : \mathcal{R} \to \mathbb{N}$ are algebra homomorphisms.

(ii) Let $\Delta = \{(n,n) \mid n \in \mathbb{N}\}$. Use the initiality of $\mathbb{N}$ to show:

$$(*) \quad \text{if } \mathcal{R} \text{ is a congruence relation then} \quad \Delta \subseteq \mathcal{R}$$

(iii) Show that $(*)$ is equivalent to the principle of mathematical induction, which says, for any predicate $P \subseteq \mathbb{N}$,

$$\text{if } P(0) \text{ and } \forall n \in \mathbb{N} : \ P(n) \Rightarrow P(n+1), \text{ then } \forall n \in \mathbb{N} : P(n)$$

(iv) Note that the characterisation of congruence in (i) above is, in a precise sense, dual to the definition of T-bisimulation, for the functor $T(X) = 1 + X$.

Exercise 1.10.6 Use finality to define a function $\mathsf{merge3} : (A^{\mathbb{N}})^3 \to A^{\mathbb{N}}$ such that

$$\mathsf{merge3}(\sigma, \tau, \rho) = (\sigma(0), \tau(0), \rho(0), \sigma(1), \tau(1), \rho(1), \ldots)$$

Next use $\mathsf{merge3}$ to define the function $\mathsf{merge}_{2,1}$ at the end of Section 1.3.

Exercise 1.10.7 Show that the singleton set $1 = \{*\}$ is (the carrier of) a final coalgebra for the functor $T(X) = X$. Show that it is also a final coalgebra for the functor $T(X) = X^A$, with A an arbitrary non-empty set.

Exercise 1.10.8 Is it possible to define the factorial function $F : \mathbb{N} \to \mathbb{N}$ given by $F(n) = n!$ by initiality? Hint: define by initiality a function $G : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ such that $\pi_1 \circ G = S$, the successor function, and $\pi_2 \circ G = F$.

Exercise 1.10.9 Let $2 = \{0, 1\}$ and let A be an arbitrary non-empty set. We consider $T(X) = 2 \times X^A$. We view a T-coalgebra $(o, n) : X \to 2 \times X^A$ as a deterministic automaton with transitions

$$x \xrightarrow{a} y \quad \Longleftrightarrow \quad n(x)(a) = y$$

A state x is accepting (final) iff $o(x) = 1$. Consider the set

$$\mathcal{P}(A^\star) = \{L \mid L \subseteq A^\star\}$$

of languages over A. Define for $L \subseteq A^\star$

$$o(L) = 1 \iff \langle\rangle \in L$$

$$n(L)(a) = \{w \in A^\star \mid \mathsf{cons}(a, w) \in L\}$$

Now show that $(\mathcal{P}(A^\star), (o, n))$ is a final T-coalgebra. Hint: show that there is a unique homomorphism from any T-coalgebra $(o, n) : X \to 2 \times X^A$ into $(\mathcal{P}(A^\star), (o, n))$ mapping any state $x \in X$ to the language that it accepts.

Exercise 1.10.10 Let $T(X) = 2 \times X^A$, as in Exercise 1.10.9, and consider a T-coalgebra $(o, n) : X \to 2 \times X^A$. Show that if x and x' in X are T-bisimilar then they are mapped by the final homomorphism to the same language. Conclude from this observation that T-bisimilarity is language equivalence.

Exercise 1.10.11 Let A be an arbitrary non-empty set. We view a $(-)^A$-coalgebra $n : X \to X^A$ as a deterministic automaton with transitions

$$x \xrightarrow{a} y \iff n(x)(a) = y$$

For $x \in X$ we define $x_\varepsilon = x$ and

$$x_{w \cdot a} = n(x_w)(a)$$

The set $A^\star$ carries a $(-)^A$-coalgebra structure $\gamma : A^\star \to (A^\star)^A$ given by

$$\gamma(w)(a) = w \cdot a$$

Show that for any $(-)^A$-coalgebra $n : X \to X^A$ with initial state $x_o : 1 \to X$ there exists a unique $(-)^A$-coalgebra homomorphism $r : (A^\star, \gamma) \to (X, n)$ s.t.

$$r(w) = (x_0)_w$$

(The function r could be called the *reachability* map.)

Exercise 1.10.12 Use finality to define, for any function $f : A \to A$, a function $\mathit{iterate}_f : A \to A^{\mathbb{N}}$ satisfying

$$\mathit{iterate}_f(a) = (a,\ f(a),\ f \circ f(a),\ \ldots)$$

Exercise 1.10.13 Show by the coinduction proof principle (cpp) that

$$\mathsf{even} \circ \mathit{iterate}_f = \mathit{iterate}_{f \circ f}$$

where even is defined in equation (1.3) of Section 1.3.

Exercise 1.10.14 Prove by the coinduction proof principle (cpp) that

$$\mathsf{odd}(\mathsf{merge}(\sigma, \tau)) = \tau$$

for all $\sigma, \tau \in A^{\mathbb{N}}$. (See Section 1.3 for the definitions of odd and merge.)

Exercise 1.10.15 For a stream $\sigma \in A^{\mathbb{N}}$ we define

$$\sigma(0) \equiv \mathsf{head}(\sigma)$$

$$\sigma' \equiv \mathsf{tail}(\sigma)$$

and call these the *initial value* and the *stream derivative* of σ. Let $a \in A$ and consider the following *stream differential equation*:

$$\sigma(0) = a \qquad \sigma' = \sigma$$

Compute the unique solution of this equation. For $a, b \in A$, compute the solution of the following system of equations:

$$\sigma(0) = a \qquad \sigma' = \tau$$

$$\tau(0) = b \qquad \tau' = \sigma$$

Exercise 1.10.16 Which function $f : A^{\mathbb{N}} \to A^{\mathbb{N}}$ is the solution of the following (functional) stream differential equation?:

$$f(\sigma)(0) = \sigma(0)$$

$$f(\sigma)' = f(\sigma'')$$

Exercise 1.10.17 The following system of stream differential equations (uniquely) defines two functions $\oplus, \otimes : \mathbb{N}^{\mathbb{N}} \times \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$:

$$(\sigma \oplus \tau)(0) = \sigma(0) + \tau(0) \qquad (\sigma \oplus \tau)' = \sigma' \oplus \tau'$$

$$(\sigma \otimes \tau)(0) = \sigma(0) \times \tau(0) \qquad (\sigma \otimes \tau)' = (\sigma' \otimes \tau) \oplus (\sigma \otimes \tau')$$

Show that for all $n \geq 0$:

$$(\sigma \oplus \tau)(n) = \sigma(n) + \tau(n)$$

$$(\sigma \otimes \tau)(n) = \sum_{k=0}^{n} \binom{n}{k} \times \sigma(k) \times \tau(n-k)$$

Exercise 1.10.18 Prove by the coinduction proof principle (cpp) that

$$(P + Q); R = (P; R) + (Q; R)$$

for all processes $P, Q, R \in \Pi$.

Bibliography

[Adá03] J. Adámek. On final coalgebras of continuous functors. *Theor. Comp. Sci.*, 294:3–29, 2003.

[AM75] M.A. Arbib and E.G. Manes. *Arrows, Structures and Functors. The Categorical Imperative.* Academic Press, New York, 1975.

[AM82] M.A. Arbib and E.G. Manes. Parametrized data types do not need highly constrained parameters. *Inf. & Contr.*, 52:139–158, 1982.

[AM89] P. Aczel and N. Mendler. A final coalgebra theorem. In D.H. Pitt, A. Poigné, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, number 389 in Lect. Notes Comp. Sci., pages 357–365. Springer, Berlin, 1989.

[Awo06] S. Awodey. *Category Theory.* Oxford University Press, 2006.

[Bar93] M. Barr. Terminal coalgebras in well-founded set theory. *Theor. Comp. Sci.*, 114(2):299–315, 1993. Corrigendum in *Theor. Comp. Sci.* 124:189–192, 1994.

[Bar03] F. Bartels. Generalised coinduction. *Math. Struct. in Comp. Sci.*, 13(2):321–348, 2003.

[BBRS09] F. Bonchi, M.M. Bonsangue, J.J.M.M. Rutten, and A. Silva. Deriving syntax and axioms for quantitative regular behaviours. In *International Conference on Concurrency Theory (CONCUR 2009)*, Lect. Notes Comp. Sci. Springer, Berlin, 2009.

[BD94] R. Burstall and R. Diaconescu. Hiding and behaviour: an institutional approach. In A.W. Roscoe, editor, *A Classical Mind. Essays in honour of C.A.R. Hoare*, pages 75–92. Prentice Hall, 1994.

[BM96] J. Barwise and L.S. Moss. *Vicious Circles: On the Mathematics of Non-wellfounded Phenomena.* CSLI Lecture Notes 60, Stanford, 1996.

[Bor94] F. Borceux. *Handbook of Categorical Algebra*, volume 50, 51 and 52 of *Encyclopedia of Mathematics.* Cambridge Univ. Press, 1994.

[BRS09] M.M. Bonsangue, J.J.M.M. Rutten, and A. Silva. Algebras for Kripke polynomial coalgebras. In *Logic in Computer Science.* IEEE, Computer Science Press, 2009.

[BV96] J.W. de Bakker and E. Vink. *Control Flow Semantics.* MIT Press, Cambridge, MA, 1996.

[BW90] M. Barr and Ch. Wells. *Category Theory for Computing Science.* Prentice Hall, 1990.

[CHL03] D. Cancila, F. Honsell, and M. Lenisa. Generalized coiteration schemata. In H.P. Gumm, editor, *Coalgebraic Methods in Computer Science*, number 82(1) in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2003. `www.sciencedirect.com/science/journal/15710661`.

[Coh81] P.M. Cohn. *Universal Algebra*, volume 6 of *Mathematics and its Applications*. D. Reidel Publ. Comp., 1981.

[CP07] C. Cîrstea and D. Pattinson. Modular proof systems for coalgebraic logics. *Theor. Comp. Sci.*, 388:83–108, 2007.

[Cro93] R.L. Crole. *Categories for Types*. Cambridge Mathematical Textbooks. Cambridge Univ. Press, 1993.

[CS95] J.R.B. Cockett and D. Spencer. Strong categorical datatypes II: A term logic for categorical programming. *Theor. Comp. Sci.*, 139:69–113, 1995.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification I: Equations and Initial Semantics*. Number 6 in EATCS Monographs. Springer, Berlin, 1985.

[Fio96] M.P. Fiore. A coinduction principle for recursive data types based on bisimulation. *Inf. & Comp.*, 127(2):186–198, 1996.

[GD94] J.A. Goguen and R. Diaconescu. Towards an algebraic semantics for the object paradigm. In H. Ehrig and F. Orejas, editors, *Recent Trends in Data Type Specification*, number 785 in Lect. Notes Comp. Sci., pages 1–29. Springer, Berlin, 1994.

[GGM76] V. Giarrantana, F. Gimona, and U. Montanari. Observability concepts in abstract data specifications. In A. Mazurkiewicz, editor, *Mathematical Foundations of Computer Science*, number 45 in Lect. Notes Comp. Sci., pages 576–587. Springer, Berlin, 1976.

[GM82] J.A Goguen and J. Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In M. Nielsen and E.M. Schmidt, editors, *International Colloquium on Automata, Languages and Programming*, number 140 in Lect. Notes Comp. Sci., pages 263–281. Springer, Berlin, 1982.

[GM93] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A theorem proing environment for higher order logic*. Cambridge Univ. Press, 1993.

[GM94] J.A. Goguen and G. Malcolm. Proof of correctness of object representations. In A.W. Roscoe, editor, *A Classical Mind. Essays in honour of C.A.R. Hoare*, pages 119–142. Prentice Hall, 1994.

[GM96] J.A. Goguen and G. Malcolm. An extended abstract of a hidden agenda. In J. Meystel, A. Meystel, and R. Quintero, editors, *Proceedings of the Conference on Intelligent Systems: A Semiotic Perspective*, pages 159–167. Nat. Inst. Stand. & Techn., 1996.

[GTW78] J.A. Goguen, J. Thatcher, and E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodoloy*, pages 80–149. Prentice Hall, 1978.

[Gum99] H.P. Gumm. Elements of the general theory of coalgebras. Notes of lectures given at LUATCS'99: Logic, Universal Algebra, Theoretical Computer Science, Johannesburg. Available as `www.mathematik.uni-marburg.de/~gumm/Papers/Luatcs.ps`, 1999.

[HJ98] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. & Comp.*, 145:107–152, 1998.

[HJ04] J. Hughes and B. Jacobs. Simulations in coalgebra. *Theor. Comp. Sci.*, 327(1-2):71–108, 2004.

[HJ05] I. Hasuo and B. Jacobs. Context-free languages via coalgebraic trace semantics. In J.L. Fiadeiro, N. Harman, M. Roggenbach, and J. Rutten, editors, *Algebra and Coalgebra in Computer Science (CALCO'05)*, number 3629 in Lect. Notes Comp. Sci., pages 213–231. Springer, Berlin, 2005.

[HJS07] I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace theory. *Logical Methods in Comp. Sci.*, 3(4:11), 2007.

[HL95] F. Honsell and M. Lenisa. Final semantics for untyped λ-calculus. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Typed Lambda Calculi and Applications*, number 902 in Lect. Notes Comp. Sci., pages 249–265. Springer, Berlin, 1995.

[HP95] M. Hofmann and B.C. Pierce. A unifying type-theoretic framework for objects. *Journ. Funct. Progr.*, 5(4):593–635, 1995.

[Jac95] B. Jacobs. Parameters and parametrization in specification using distributive categories. *Fund. Informaticae*, 24(3):209–250, 1995.

[Jac96a] B. Jacobs. Inheritance and cofree constructions. In P. Cointe, editor, *European Conference on Object-Oriented Programming*, number 1098 in Lect. Notes Comp. Sci., pages 210–231. Springer, Berlin, 1996.

[Jac96b] B. Jacobs. Objects and classes, co-algebraically. In B. Freitag, C.B. Jones, C. Lengauer, and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer Acad. Publ., 1996.

[Jac06] B. Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In K. Futatsugi, J.-P. Jouannaud, and J. Meseguer, editors, *Algebra, Meaning and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, number 4060 in Lect. Notes Comp. Sci., pages 375–404. Springer, Berlin, 2006.

[Jac08] B. Jacobs. Coalgebraic trace semantics for combined possibilitistic and probabilistic systems. In J. Adámek and C. Kupke, editors, *Coalgebraic Methods in Computer Science*, volume 203(5) of *Elect. Notes in Theor. Comp. Sci.*, pages 131–152. Elsevier, Amsterdam, 2008.

[JNW96] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Inf. & Comp.*, 127(2):164–185, 1996.

[Kam83] S. Kamin. Final data types and their specification. *ACM Trans. on Progr. Lang. and Systems*, 5(1):97–123, 1983.

[Kli07] B. Klin. Coalgebraic modal logic beyond sets. In M. Fiore, editor, *Mathematical Foundations of Programming Semantics*, number 173 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2007.

[Kur01] A. Kurz. Specifying coalgebras with modal logic. *Theor. Comp. Sci.*, 260(1-2):119–138, 2001.

[Kur06] A. Kurz. Coalgebras and their logics. *SIGACT News Logic Column*, 37(2):57–77, 2006. Available from `http://arxiv.org/abs/cs.LO/0605128`.

[Lan71] S. Mac Lane. *Categories for the Working Mathematician*. Springer, Berlin, 1971.

[LS81] D.J. Lehmann and M.B. Smyth. Algebraic specification of data types: A synthetic approach. *Math. Systems Theory*, 14:97–139, 1981.

[LS86] J. Lambek and P.J. Scott. *Introduction to higher order Categorical Logic*. Number 7 in Cambridge Studies in Advanced Mathematics. Cambridge Univ. Press, 1986.

[MA86] E.G. Manes and M.A. Arbib. *Algebraic Appoaches to Program Semantics.* Texts and Monogr. in Comp. Sci.,. Springer, Berlin, 1986.

[Mal96] G. Malcolm. Behavioural equivalence, bisimulation and minimal realisation. In M. Haveraaen, O. Owe, and O.J. Dahl, editors, *Recent Trends in Data Type Specification*, number 1130 in Lect. Notes Comp. Sci., pages 359–378. Springer, Berlin, 1996.

[Man74] E.G. Manes. *Algebraic Theories.* Springer, Berlin, 1974.

[Mel89] T.F. Melham. Automating recursive type definitions in higher order logic. In G. Birtwistle and P.A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, Lect. Notes Comp. Sci., pages 341–386. Springer, Berlin, 1989.

[Mil89] R. Milner. *A Calculus of Communicating Systems.* Lect. Notes Comp. Sci. Springer, Berlin, 1989.

[MT92] K. Meinke and J.V. Tucker. Universal algebra. In S. Abramsky, Dov M. Gabbai, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 189–411. Oxford Univ. Press, 1992.

[ORR$^+$96] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Computer Aided Verification*, number 1102 in Lect. Notes Comp. Sci., pages 411–414. Springer, Berlin, 1996.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI Conference on Theoretical Computer Science*, number 104 in Lect. Notes Comp. Sci., pages 15–32. Springer, Berlin, 1981.

[Pau90] L.C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and computer science*, pages 361–386. Academic Press, London, 1990. The APIC series, vol. 31.

[Pau97] L.C. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *Journ. of Logic and Computation*, 7:175–204, 1997.

[Pie91] B.C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, Cambridge, MA, 1991.

[Pit94] A.M. Pitts. A co-induction principle for recursively defined domains. *Theor. Comp. Sci.*, 124(2):195–219, 1994.

[Pit96] A.M. Pitts. Relational properties of domains. *Inf. & Comp.*, 127(2):66–90, 1996.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus Univ., 1981.

[PM93] Ch. Paulin-Mohring. Inductive definitions in the system Coq. Rules and properties. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, number 664 in Lect. Notes Comp. Sci., pages 328–345. Springer, Berlin, 1993.

[PP02] D. Pavlović and V. Pratt. The continuum as a final coalgebra. *Theor. Comp. Sci.*, 280 (1-2):105–122, 2002.

[Rei95] H. Reichel. An approach to object semantics based on terminal co-algebras. *Math. Struct. in Comp. Sci.*, 5:129–152, 1995.

[Rou96] W.C. Rounds. Feature logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language.* Elsevier, 1996.

[RT94] J. Rutten and D. Turi. Initial algebra and final coalgebra semantics for concurrency. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *A Decade* *of Concurrency*, number 803 in Lect. Notes Comp. Sci., pages 530–582. Springer, Berlin, 1994.

[Rut00] J. Rutten. Universal coalgebra: a theory of systems. *Theor. Comp. Sci.*, 249:3–80, 2000.

[Rut03] J. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comp. Sci.*, 308:1–53, 2003.

[SP82] M.B. Smyth and G.D. Plotkin. The category theoretic solution of recursive domain equations. *SIAM Journ. Comput.*, 11:761–783, 1982.

[SR07] A. Silva and J. Rutten. Behavioural differential equations and coinduction for binary trees. In D. Leivant and R. de Queiroz, editors, *Logic, Language, Information and Computation, 14th International Workshop, WoLLIC 2007*, number 4576 in Lect. Notes Comp. Sci., pages 322–336. Springer, Berlin, 2007.

[TP97] D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Logic in Computer Science*, pages 280–291. IEEE, Computer Science Press, 1997.

[Tur96] D. Turi. *Functorial operational semantics and its denotational dual.* PhD thesis, Free Univ. Amsterdam, 1996.

[Wal91] R.F.C. Walters. *Categories and Computer Science.* Carslaw Publications, Sydney, 1991. Also available as: Cambridge Computer Science Text 28, 1992.

[Wec92] W. Wechler. *Universal Algebra for Computer Scientists.* Number 25 in EATCS Monographs. Springer, Berlin, 1992.

[Wir90] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 673–788. Elsevier/MIT Press, 1990.