Introduction to Coalgebra.
Towards Mathematics of States and Observations

Bart Jacobs

Institute for Computing and Information Sciences,

Radboud University Nijmegen,

P.O. Box 9010, 6500 GL Nijmegen, The Netherlands.

bart@cs.ru.nl http://www.cs.ru.nl/~bart

Draft Copy.

Comments / bugs / improvements *etc.* are welcome at bart@cs.ru.nl

(Please check the latest version on the web first,
to see if the issue you wish to report has already been adderessed)

Version 1.00, 22nd April 2005

Preface

Mathematics is about the formal structures underlying counting, measuring, transforming *etc*. It has developed fundamental notions like number systems, groups, vector spaces, see *e.g.* [168], and has studied their properties. In more recent decades also "dynamical" features have become a subject of research. The emergence of computers has contributed to this development. Typically, dynamics involves a "state of affairs", which can possibly be observed and modified. For instance, the contents of a tape of a Turing machine contribute to its state. Such a machine may thus have many possible states, and can move from one state to another. Also, the combined contents of all memory cells of a computer can be understood as the computers state. A user can observe part of this state via the screen (or via the printer), and modify this state by typing commands. In reaction, the computer can display certain behaviour. Describing the behaviour of such a computer system is a non-trival matter. However, formal descriptions of such complicated systems are needed if we wish to reason formally about their behaviour. Such reasoning is required for the correcteness or security of these systems. It involves a specification describing the required behaviour, together with a correctness proof demonstrating that a given implementation satisfies the specification.

Mathematicians and computer scientists have introduced various formal structures to capture the essence of state-based dynamics, such as automata (in various forms), transitions systems, Petri nets, event systems, *etc*. The area of coalgebras has emerged with a unifying claim. It aims to be the mathematics of computational dynamics. It combines notions and ideas from the mathematical theory of dynamical systems and from the theory of state-based computation. The area of coalgebra is still in its infancy, but promises a perspective on uniting, say, the theory of differential equations with automata and process theory, by providing an appropriate semantical basis with associated logic. The theory of coalgebras may be seen as one of the original contributions stemming from the area of theoretical computer science. The span of applications of coalgebras is still fairly limited, but may in the future be extended to include dynamical phenomena in areas like physics, biology or economics—based for instance on the claim of Adleman (the father of DNA-computing) that biological life can be equated with computation [12]; or on [186] which gives a coalgebraic description of type spaces used in economics [110].

Coalgebras are of surprising simplicity. They consist of state space, or set of states, say X, together with a structure map of the form $X \to F(X)$. The symbol F describes some expression involving X (a functor), capturing the possible outcomes of the structure map applied to a state. The map $X \to F(X)$ captures the dynamics in the form of a function acting on states. For instance, one can have $F(X) = \mathcal{P}(X)$ for non-deterministic computation $X \to \mathcal{P}(X)$, or $F(X) = \{\bot\} \cup X$ for possibly non-terminating computations $X \to \{\bot\} \cup X$. At this level of generality, algebras are described as the duals of coalgebras (or the other way round), namely as maps of the form $F(X) \to X$.

Computer science is about generated behaviour

What is the essence of computing? What is the topic of the discipline of computer science? Answers that are often heard are 'data processing' or 'symbol manipulation'. Here

we follow a more behaviouristic approach and describe the subject of computer science as *generated behaviour*. This is the behaviour that can be observed on the outside of a computer, for instance via a screen or printer. It arises in interaction with the environment, as a result of the computer executing instructions, layed down in a computer program. The aim of computer programming is to make a computer do certain things, *i.e.* to generate behaviour. By executing a program a computer displays behaviour that is ultimately produced by humans, as programmers.

This behaviouristic view allows us to understand the relation between computer science and the natural sciences: biology is about "spontaneous" behaviour, and physics concentrates on lifeless natural phenomena, without autonomous behaviour. Behaviour of a system in biology or physics is often described as evolution, where evolutions in physics are transformational changes according to the laws of physics. Evolutions in biology seem to lack inherent directionality and predictability [98]. Does this mean that behaviour is deterministic in (classical) physics, and non-deterministic in biology? And that coalgebras of corresponding kinds capture the situation? At this stage the coalgebraic theory of modeling has not yet demonstrated its usefulness in those areas. Therefore this text concentrates on coalgebras in mathematics and computer science.

The behaviouristic view does help in answering questions like: can a computer think? Or: does a computer feel pain? All a computer can do is display thinking behaviour, or pain behaviour, and that is it. But it is good enough in interactions—think of the famous Turing test—because in the end we never know for sure if other people actually feel pain. We only see pain behaviour, and are conditioned to associate such behaviour with certain internal states. But this association may not always work, for instance not in a different culture: in Japan it is common to touch one's ear after burning a finger; for Europeans this is non-standard pain behaviour. This issue of external behaviour versus internal states is nicely demonstrated in [177] where it turns out to be surprisingly difficult for a human to kill a "Mark III Beast" robot, once it starts displaying desparate survival behaviour with corresponding sounds, so that people easily attribute feelings to the machine and start to feel pity.

These wide-ranging considerations form the background for a theory about computational behaviour in which the relation between observables and internal states is of central importance.

The generated behaviour that we claim to be the subject of computer science arises by a computer executing a program according to strict operational rules. The behaviour is typically observed via the computer's input & output (I/O). More technically, the program can be understood as an element in an inductively defined set P of terms. This set forms a suitable (initial) algebra $F(P) \to P$, where the expression (or functor) F captures the signature of the operations for forming programs. The operational rules for the behaviour of programs are described by a coalgebra $P \to G(P)$, where the functor G captures the kind of behaviour that can be displayed—such as deterministic, or with exceptions. In abstract form, generated computer behaviour amounts to the repeated evaluation of an (inductively defined) coalgebra structure on an algebra of terms. Hence the algebras and coalgebras that are studied systematically in this text form the basic structures at the heart of computer science.

One of the big challenges of computer science is to develop techniques for effectively establishing properties of generated behaviour. Often such properties are formulated positively as wanted, functional behaviour. But these properties may also be negative, like in computer security, where unwanted behaviour must be excluded. However, an appropriate logical view about program properties within the combined algebraic/coalgebraic setting has not been fully elaborated yet.

Algebras and coalgebras

The duality with algebras forms a source of inspiration and of opposition: there is a "hate-love" relationship between algebra and coalgebra. First, there is a fundamental divide. Think of the difference between an inductively defined data type in a functional programming language (an algebra) and a class in an object-oriented programming language (a coalgebra). The data type is completely determined by its "constructors": algebraic operations of the form $F(X) \to X$ going *into* the data type. The class however involves an internal state, given by the values of all the public and private fields of the class. This state can be observed (via the public fields) and can be modified (via the public methods). These operations of a class act on a state (or object) and are naturally described as "destructors" pointing *out of* the class: they are of the coalgebraic form $X \to F(X)$.

Next, besides these differences between algebras and coalgebras there are also many correspondences, analogies, and dualities, for instance between bisimulations and congruences, or between initiality and finality. Whenever possible, these connections will be made explicit and will be exploited in the course of this work.

As already mentioned, ultimately, stripped to its bare minimum, a programming language involves both a coalgebra and an algebra. A program is an element of the algebra that arises (as so-called initial algebra) from the programming language that is being used. Each language construct corresponds to certain dynamics, captured via a coalgebra. The program's behaviour is thus described by a coalgebra acting on the state space of the computer. This is the view underlying the so-called structural operational semantics. Coalgebraic behaviour is generated by an algebraic program. This is a simple, clear and appealing view. It turns out that in such situations one needs a certain level of compatibility between the algebras and coalgebras involved. It is expressed in terms of so-called distributive laws connecting algebra-coalgebra pairs. These laws appear towards the end of this text.

Coalgebras have a black box state space

Coalgebra is thus the study of states and their operations and properties. The set of states is best seen as a black box, to which one has limited access—like with the states of a computer mentioned above. As already mentioned, the tension between what is actually inside and what can be observed externally is at the heart of the theory of coalgebras. Such tension also arises for instance in quantum mechanics where the relation between observables and states is an issue. Similarly, it is an essential element of cryptography that parts of data are not observable—via encryption or hashing. In a coalgebra it may very well be the case that two states are internally different, but are indistinguishable as far as one can see with the available operations. In that case one calls the two states *bisimilar*. Bisimilarity is indeed one of the fundamental notions of the theory of coalgebras, see Chapter 3. Also important are *invariant* properties of states: once such a property holds, it continues to hold no matter which of the available operations is applied, see Chapter 4. Safety properties of systems are typically expressed as invariants. Finally, specifications of the behaviour of systems are conveniently expressed using assertions and modal operators like: for all direct successor states (nexttime), for all future states (henceforth), for some future state (eventually), see Chapter 5. This text describes these basic elements of the theory of coalgebras—bisimilarity, invariants and assertions. It is meant as an introduction to this new and fascinating field within theoretical computer science. The text is too limited in both size and aims to justify the grand unifying claims mentioned above. But hopefully, it does inspire and generate much further research in the area.

Brief historical perspective

Coalgebra does not come out of the blue. Below we shall sketch several, relatively independent, developments during the last few decades that appeared to have a common

coalgebraic basis, and that have contributed to the area of coalgebra as it stands today. This short sketch is of course far from complete.

1. **The categorical approach to mathematical system theory.** During the 1970s Arbib, Manes and Goguen analysed Kalman's [153] work on linear dynamical systems, in relation to automata theory. They realised that linearity does not really play a role in Kalman's famous results about minimal realisation and duality, and that these results could be reformulated and proved more abstractly using elementary categorical constructions. Their aim was "to place sequential machines and control systems in a unified framework" (abstract of [20]), by developing a notion of "machine in a category" (see also [8]). This led to general notions of state, behaviour, reachability, observability, and realisation of behaviour. However, the notion of coalgebra did not emerge explicitly in this approach, probably because the setting of modules and vector spaces from which this work arose provided too little categorical infrastructure (especially: no Cartesian closure) to express these results purely coalgebraically.

2. **Non-well-founded sets.** Aczel [4] formed a next crucial step with his special set theory that allows infinitely descending $\in$-chains, because it used coalgebraic terminology right from the beginning. The development of this theory was motivated by the desire to provide meaning to Milner's theory CCS of concurrent processes with potentially infinite behaviour. Therefore, the notion of bisimulation from process theory played a crucial role. An important contribution of Aczel is that he showed how to treat bisimulation in a coalgebraic setting, especially by establishing the first link between proofs by bisimulations and finality of coalgebras, see also [7, 5].

3. **Data types of infinite objects.** The first systematic approach to data types in computing [93] relied on initiality of algebras. The elements of such algebraic structures are finitely generated objects. However, many data types of interest in computing (and mathematics) consist of infinite objects, like infinite lists or trees (or even real numbers). The use of (final) coalgebras in [237, 21, 107, 193] to capture such structures provided a next important step.

4. **Initial and final semantics.** In the semantics of program and process languages it appeared that the relevant semantical domains carry the structure of a final coalgebra (sometimes in combination with initial algebra structure [80, 72]). Especially in the metric space based tradition (see *e.g.* [26]) this insight was combined with Aczel's techniques by Rutten and Turi. It culminated in the recognition that "compatible" algebra-coalgebra pairs (called bialgebras) are highly relevant structures, described via distributive laws. The basic observation of [232, 231], further elaborated in [33], is that such laws correspond to specification formats for operational rules on (inductively defined) programs. These bialgebras satisfy elementary properties like: observational equivalence (*i.e.* bisimulation wrt. the coalgebra) is a congruence (wrt. the algebra).

5. **Behavioural approaches in specification.** Reichel [204] was the first to use so-called behavioural validity of equations in the specification of algebraic structures that are computationally relevant. The basic idea is to divide one's types (also called sorts) into 'visible' and 'hidden' ones. The latter are supposed to capture states, and are not directly accessible. Equality is only used for the "observable" elements of visible types. For elements of hidden types (or states) one uses behavioural equality instead: two elements x_1 and x_2 of hidden type are behaviourally equivalent if $t(x_1) = t(x_2)$ for each term t of visible type. This means that they are equal as far as can be observed. The idea is further elaborated in what has become known as hidden algebra [92], see for instance also [84, 223, 39], and has been applied to describe classes in object-oriented programming languages, which have an encapsulated state space. But it was later realised that behavioural equality is essentially bisimilarity in a coalgebraic context (see *e.g.* [172]), and it was again Reichel [206] who first used coalgebras for the semantics of object-oriented languages. Later on they have been applied also to actual programming languages like Java [141].

6. **Modal logic.** A more recent development is the connection between coalgebras and modal logics. In general, such logics qualify the truth conditions of statements, concerning knowledge, belief and time. In computer science such logics are used to reason about the way programs behave, and to express dynamical properties of transitions between states. Temporal logic is a part of modal logic which is particularly suitable for reasoning about (reactive) state-based systems, as argued for example in [200, 201], via its nexttime and lasttime operators. Since coalgebras give abstract formalisations of such state-based systems one expects a connection. It was Moss [183] who first associated a suitable modal logic to coalgebras—which inspired much subsequent work [209, 211, 165, 123, 136, 190, 163]. The idea is that the role of equational formulas in algebra is played by modal formulas in coalgebra.

Position of this text

There are several recent texts presenting a synthesis of several of the developments in the area of coalgebra [144, 233, 100, 216, 164, 191]. This text is a first systematic presentation of the subject in the form of a book. Key phrases are: coalgebras are general dynamical systems, final coalgebras describe behaviour of such systems (often as infinite objects) in which states and observations coincide, bisimilarity expresses observational indistinguishability, the natural logic of coalgebras is modal logic, *etc*.

During the last decade a "coalgebraic community" has emerged, centered around the workshops *Coalgebraic Methods in Computer Science*, see the proceedings [139, 145, 207, 57, 185, 101, 11] and associated special journal issues [140, 146, 58, 102]. This text is specifically not focused on that community, but tries to reach a wider audience. This means that the emphasis lies on explaining the theory via concrete examples, and on motivation rather than on generality and abstraction.

Coalgebra and category theory

The field of coalgebra requires the theory of categories already in the definition of the notion of coalgebra itself—since it requires the concept of a functor. However, the reader is not assumed to know category theory: in this text the intention is not to describe the theory of coalgebras in its highest form of generality, making systematic use of category theory right from the beginning. After all, this is only an introduction. Rather, the text starts from concrete examples and introduces the basics of category theory as it proceeds. Categories will thus be introduced gradually, without making it a proper subject matter. Hopefully, readers unfamiliar with category theory can thus pick up the basics along the way, seeing directly how it is used. Anyway, most of the examples that are discussed live in the familiar standard setting of sets and functions, so that it should be relatively easy to see the underlying categorical structures in a concrete setting. Thus, more or less familiar set-theoretic language is used most of the time, but with a perspective on the greater generality offered by the theory of categories. In this way we hope to serve the readers without background in category theory, and at the same time offer the more experienced *cognoscienti* an idea of what is going on at a more abstract level—which they can find to a limited extent in the exercises, but to a greater extent in the literature. Clearly, this is a compromise which runs the risk of satisfying no-one: the description may be too abstract for some, and too concrete for others. The hope is that it does have something to offer for everyone.

Often the theory of categories is seen as a very abstract part of mathematics, that is not very accessible. However, it is essential in this text, for several good reasons.

1. It greatly helps to properly organise the relevant material on coalgebras.
2. Only by using categorical language the duality between coalgebra and algebra can be fully seen—and exploited.
3. Almost all of the literature on coalgebra uses category theory in one way or another. Therefore, an introductory text that wishes to properly prepare the reader for further study cannot avoid the language of categories.

Probably the most controversial aspect of this text within the coalgebraic / categorical community is its restriction to so-called polynomial functors, and its emphasis on the associated operations of predicate and relation lifting. Again, this is motivated by our wish to produce an introduction that is accessible to non-specialists. Certainly, the general perspective is always right around the corner, and will hopefully be appreciated once this more introductory material has been digested.

In the end, we think that coalgebras form a very basic and natural mathematical concept, and that their identification is real step forward. Many people seem to be using coalgebras in various situations, without being aware of it. Hopefully this text can make them aware, and can contribute to a better understanding and exploitation of these situations. And hopefully many more such application areas will be identified, further enriching the theory of coalgebras.

Intended audience

This text is written for everyone with an interest in the mathematical aspects of computational behaviour. This probably includes primarily mathematicians, logicians and (theoretical) computer scientists, but hopefully also an audience with a different background such as for instance mathematical physics or biology, or even economics. A basic level of mathematical maturity is assumed, for instance via familiarity with elementary set theory and logic (and its notation). The examples in the text are taken from various areas. Each section is accompagnied by a series of exercises, to facilitate teaching—typically at a late bachelor or early master level.

Acknowledgements

[To be written.]

Contents

Preface **iii**

1 Motivation **1**
1.1 Naturalness of coalgebraic representations . . . 2
1.2 The power of the coinduction . . . 4
1.3 Generality of temporal logic of coalgebras . . . 13
1.3.1 Temporal operators for sequences . . . 13
1.3.2 Temporal operators for classes . . . 16
1.4 Abstractness of the coalgebraic notions . . . 18

2 Preliminaries on coalgebras and algebras **23**
2.1 Constructions on sets . . . 23
2.2 Polynomial functors and their coalgebras . . . 31
2.2.1 Statements and sequences . . . 32
2.2.2 Trees . . . 33
2.2.3 Deterministic automata . . . 34
2.2.4 Non-deterministic automata and transition systems . . . 36
2.2.5 Context-free grammars . . . 37
2.2.6 Non-well-founded sets . . . 38
2.3 Final coalgebras . . . 40
2.3.1 Beyond sets . . . 45
2.4 Algebras . . . 46
2.4.1 Bialgebras . . . 51
2.4.2 Dialgebras . . . 51
2.4.3 Hidden algebras . . . 51
2.4.4 Coalgebras as algebras . . . 52
2.5 Adjunctions, cofree coalgebras, behaviour-realisation . . . 53

3 Bisimulations **65**
3.1 Relation lifting, bisimulations and congruences . . . 65
3.2 Properties of bisimulations . . . 71
3.3 Bisimulations as spans . . . 78
3.3.1 Comparing definitions of bisimulation . . . 81
3.4 Bisimulations and the coinduction proof principle . . . 86
3.5 Process semantics . . . 91
3.5.1 Process descriptions . . . 92
3.5.2 A simple process algebra . . . 95

4 Invariants **99**
4.1 Predicate lifting . . . 99
4.1.1 Predicate lowering as liftings left adjoint . . . 102
4.2 Invariants . . . 105

4.2.1 Greatest invariants and limits of coalgebras . . . 108
4.3 Temporal logic of coalgebras . . . 113
4.3.1 Backward reasoning . . . 121
4.4 Existence of final coalgebras . . . 124
4.4.1 Final coalgebras for ω-continuous functors . . . 124
4.4.2 Final coalgebras for ω-accessible functors . . . 126
4.5 Trace semantics . . . 129

5 Assertions [Not included] 137
5.1 Arities . . . 137
5.2 Algebraic specification . . . 140
5.2.1 Monads . . . 143
5.3 Coalgebraic specification . . . 147
5.3.1 Comonads . . . 152
5.4 Modal logic of coalgebras . . . 154
5.4.1 A bounded stack specification . . . 157
5.5 Modal algebras and coalgebras . . . 160
5.6 Coalgebraic class specifications . . . 166
5.7 Solving recursive equations via finality . . . 168
5.7.1 Finite and infinite terms, and substitution . . . 168
5.7.2 Recursive equations . . . 172

6 Algebra meets coalgebra 181

References 181

Subject Index 197

Definition and Symbol Index 205

Chapter 1

Motivation

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

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

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

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

A simple example of a coalgebra is the function,

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

with state space $\mathbb{Z}$ occurring twice on the right hand side. Thus the box or type of this coalgebra is: $\boxed{(-) \times (-)}$. The transition function $n \mapsto (n-1, n+1)$ may also be written using λ-notation as $\lambda n.\, (n-1, n+1)$ or as $\lambda n \in \mathbb{Z}.\, (n-1, n+1)$.

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

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

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

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

Many more examples of coalgebras will occur throughout this text.

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

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

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

1.1 Naturalness of coalgebraic represenations

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

Our programs should thus be able to modify a state, typically via an assignment like `i = 5` in a so-called imperative programming language[2]. Such an assignment statement is interpreted as a function that turns a state x into a new, successor state x' in which the value of the identifier `i` is equal to `5`. Statements in such languages are thus described via suitable "state transformer" functions. In simplest form, ignoring input and output, they map a state to a successor state, as in:

$$S \xrightarrow{\mathsf{stat}} S$$

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

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

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

1. **Adjust the state space**. In this case one extends the state space S to a space $S_\perp \stackrel{\text{def}}{=} \{\perp\} \cup S$, where $\perp$ is a new "bottom" element not occurring in S that is especially used to signal non-termination. Statements then become functions:

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

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

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

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

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

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

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

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

We push these same ideas a bit further. In many programming languages (like Java [23]) programs may not only hang, but may also terminate "abruptly" because of an exception. An exception arises when some constraint is violated, such as a division by zero or an access `a[i]` in an array `a` which is a null-reference. Abrupt termination is fundamentallty different from non-termination: non-termination is definitive and irrevocable, whereas a program can recover from abrupt termination via a suitable exception handler that restores nornal termination. In Java this is done via a `try-catch` statement, see for instance [23, 97, 132].

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

1. **Adjust the state space**. Statements then remain endofunctions[3] on an extended state space:

$$\Big(\{\perp\} \cup S \cup (S \times E)\Big) \xrightarrow{\mathsf{stat}} \Big(\{\perp\} \cup S \cup (S \times E)\Big)$$

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

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

[2] Thus, purely functional programming languages are not included in our investigations.

[3] An endofunction is a function $A \to A$ from a set A to itself.

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

$$S \xrightarrow{\ \mathsf{stat}\ } \Big(\{\bot\} \cup S \cup (S \times E)\Big)$$

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

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

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

When such a coalgebraic representation is formalised within the typed language of a theorem prover (like in [142]), the typechecker of the theorem prover will make sure that case distinctions are made. See also [132] where Java's exception mechanism is described via such case distinctions, closely following the official language definition [97].

These examples illustrate that coalgebras as functions with structured codomains, like in (1.1), arise naturally, and that the structure of the codomain indicates the kind of computations that can be performed. This idea will be developed further, and applied to various forms of computation. For instance, non-deterministic statements may be represented via the powerset $\mathcal{P}$ as coalgebraic state transformers $S \to \mathcal{P}(S)$ with multiple result states. But there are many more such examples.

(Readers familiar with computational monads [182] may recognise similarities. Indeed, in a computational setting there is a close connection between coalgebraic and monadic representations. Briefly, the monad introduces the computational structure, like composition and extension, whereas the coalgebraic view leads to an appropriate program logic. This is elaborated for Java in [141].)

Exercises

1.1.1. (i) Prove that the composition operation ; as defined for coalgebras $S \to \{\bot\} \cup S$ is associative, *i.e.* satisfies $s_1 \,;(s_2 \,; s_3) = (s_1 \,; s_2) \,; s_3$, for all statements $s_1, s_2, s_3 \colon S \to \{\bot\} \cup S$.
Define a statement $\mathsf{skip} \colon S \to \{\bot\} \cup S$ which is a unit for composition ;, *i.e.* which satisfies $\mathsf{skip} \,; s = s = s \,; \mathsf{skip}$, for all $s \colon S \to \{\bot\} \cup S$.
(ii) Do the same for ; defined on coalgebras $S \to \{\bot\} \cup S \cup (S \times E)$.
[In both cases, statements with an associative composition operation and a unit element form a *monoid*.]

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

1.2 The power of the coinduction

In this section we shall look at sequences—or lists, or words, as they are also called. Sequences are basic data structures, both in mathematics and in computer science. One can distinguish finite sequences $\langle a_1, \ldots, a_n \rangle$ and infinite $\langle a_1, a_2, \ldots \rangle$ ones. The mathematical theory of finite sequences is well-understood, and a fundamental part of computer science,

used in many programs. Definition and reasoning with finite lists is commonly done with induction. As we shall see, infinite lists require *coinduction*. Infinite sequences can arise in computing as the observable outcomes of a program that runs forever. Also, in functional programming, they can occur as so-called lazy lists, like in the languages Haskell [41] or Clean [196].

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

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

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

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

$$\begin{array}{ccc} A^\infty & \xrightarrow{\ \mathsf{next}\ } & \{\bot\} \cup \big(A \times A^\infty\big) \\ \sigma & \longmapsto & \begin{cases} \bot & \text{if } \sigma \text{ is the empty sequence } \langle\rangle \\ (a, \sigma') & \text{if } \sigma = a \cdot \sigma' \text{ with "head" } a \in A \text{ and "tail" } \sigma' \in A^\infty \end{cases} \end{array} \tag{1.2}$$

The type of the coalgebra is thus $\boxed{\{\bot\} \cup (A \times (-))}$. The successor of a state $\sigma \in A^\infty$, if any, is its tail sequence, obtained by removing the head.

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

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

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

1.2.1. Proposition (Finality of sequences). *The coalgebra* $\mathsf{next} \colon A^\infty \to \{\bot\} \cup A \times A^\infty$ *from (1.2) is final among coalgebras of this type: for an arbitrary coalgebra* $c \colon S \to \{\bot\} \cup (A \times S)$ *on a set* S *there is a unique "behaviour" function* $\mathsf{beh}_c \colon S \to A^\infty$ *which is a homomorphism of coalgebras. That is, for each* $x \in S$*, both:*

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

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

$$\begin{array}{ccc} \{\bot\} \cup (A \times S) & \xrightarrow{\{\bot\} \cup (A \times \mathsf{beh}_c)} & \{\bot\} \cup (A \times A^\infty) \\ {\scriptstyle c}\uparrow & & \cong \uparrow{\scriptstyle \mathsf{next}} \\ S & \xrightarrow[\mathsf{beh}_c]{} & A^\infty \end{array}$$

where the function $\{\bot\} \cup (A \times \mathsf{beh}_c)$ *on top maps* $\bot$ *to* $\bot$ *and* (a, x) *to* $(a, \mathsf{beh}_c(x))$.

In the course of this chapter we shall see that a general notion of homomorphism between coalgebras (of the same type) can be defined by such commuting diagrams.

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

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

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

$$\begin{array}{rcl} c^0(x) & = & c(x) \\ c^{n+1}(x) & = & \begin{cases} \bot & \text{if } c^n(x) = \bot \\ c(y) & \text{if } c^n(x) = (a, y) \end{cases} \end{array}$$

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

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

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

- If $c(x) = \bot$, then the least m with $c^m(x) = \bot$ is 0, so that $\mathsf{beh}_c(x) = \langle\rangle$, and thus also $\mathsf{next}(\mathsf{beh}_c(x)) = \bot$.
- If $c(x) = (a, x')$, then we distinguish two cases:
  - If $\forall n \in \mathbb{N}.\, c^n(x) \neq \bot$, then $\forall n \in \mathbb{N}.\, c^n(x') \neq \bot$, and $c^{i+1}(x) = c^i(x')$. Let $c^i(x') = (a_i, x_i)$, then

$$\begin{array}{rcl} \mathsf{next}(\mathsf{beh}_c(x)) & = & \mathsf{next}(\langle a, a_0, a_1, \ldots \rangle) \\ & = & (a, \langle a_0, a_1, \ldots \rangle) \\ & = & (a, \mathsf{beh}_c(x')). \end{array}$$

  - If m is least with $c^m(x) = \bot$, then $m > 0$ and $m - 1$ is the least k with $c^k(x') = \bot$. For $i < m - 1$ we have $c^{i+1}(x) = c^i(x')$, and thus by writing $c^i(x') = (a_i, x_i)$, we get as before:

$$\begin{array}{rcl} \mathsf{next}(\mathsf{beh}_c(x)) & = & \mathsf{next}(\langle a, a_0, a_1, \ldots, a_{m-2} \rangle) \\ & = & (a, \langle a_0, a_1, \ldots, a_{m-2} \rangle) \\ & = & (a, \mathsf{beh}_c(x')). \end{array}$$

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

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

Before exploiting this finality result we illustrate the behaviour function.

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

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

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

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

A little thought brings us to the following:

$$\mathsf{nextdec}(r) \;=\; \begin{cases} \bot & \text{if } r = 0 \\ (d, 10r - d) & \text{otherwise, where } d \text{ is such that } d \leq 10r < d + 1. \end{cases}$$

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

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

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

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

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

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

Thus the resulting $\mathsf{nextdec}$-behaviour on $\frac{1}{8}$ is indeed $\langle 1, 2, 5 \rangle$, *i.e.* $\mathsf{beh}_{\mathsf{nextdec}}(\frac{1}{8}) = \langle 1, 2, 5 \rangle$.

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

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

Thus $\mathsf{nextdec}$ immediately loops on $\frac{1}{9}$, and we get an infinite sequence $\langle 1, 1, 1, \ldots \rangle$ as behaviour. This completes the example.

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

- **A coinductive definition principle.** The existence aspect tells us how to obtain functions $S \to A^\infty$ into A^∞.
- **A conductive proof principle.** The uniqueness aspect tells us how to prove that two functions $f, g\colon S \to A^\infty$ are equal, namely by showing that they are both homomorphisms from one coalgebra $c\colon S \to \{\bot\} \cup (A \times S)$ to the final coalgebra $\mathsf{next}\colon A^\infty \to \{\bot\} \cup (A \times A^\infty)$.

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

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

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

$$x \not\to \quad \text{if } c(x) = \bot \qquad \text{and} \qquad x \xrightarrow{a} x' \quad \text{if } c(x) = (a, x'). \tag{1.3}$$

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

This transition notation applies in particular to the final coalgebra $\mathsf{next}\colon A^\infty \to \{\bot\} \cup (A \times A^\infty)$. In that case, for $\sigma \in A^\infty$, $\sigma \not\to$ means that the sequence σ is empty. In the second case $\sigma \xrightarrow{a} \sigma'$ expresses that the sequence σ can do an a-step to σ', and hence that $\sigma = a \cdot \sigma'$.

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

- $x \not\to \Longrightarrow \mathsf{beh}_c(x) \not\to$;
- $x \xrightarrow{a} x' \Longrightarrow \mathsf{beh}_c(x) \xrightarrow{a} \mathsf{beh}_c(x')$.

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

$$\frac{x \not\to}{\mathsf{beh}_c(x) \not\to} \qquad \frac{x \xrightarrow{a} x'}{\mathsf{beh}_c(x) \xrightarrow{a} \mathsf{beh}_c(x')} \tag{1.4}$$

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

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

Evenly listed elements from a sequence

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

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

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

Like in (1.4) above we can write these three requirements as rules:

$$\frac{\sigma \not\to}{\mathsf{evens}(\sigma) \not\to} \qquad \frac{\sigma \xrightarrow{a} \sigma' \qquad \sigma' \not\to}{\mathsf{evens}(\sigma) \xrightarrow{a} \mathsf{evens}(\sigma')} \qquad \frac{\sigma \xrightarrow{a} \sigma' \qquad \sigma' \xrightarrow{a'} \sigma''}{\mathsf{evens}(\sigma) \xrightarrow{a} \mathsf{evens}(\sigma'')} \tag{1.5}$$

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

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

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

$$\begin{array}{ccc} \{\bot\} \cup (A \times A^\infty) & \overset{\{\bot\} \cup (A \times \mathsf{beh}_e)}{\dashrightarrow} & \{\bot\} \cup (A \times A^\infty) \\ {\scriptstyle e}\uparrow & & \cong \uparrow{\scriptstyle \mathsf{next}} \\ A^\infty & \underset{\mathsf{evens} = \mathsf{beh}_e}{\dashrightarrow} & A^\infty \end{array}$$

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

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

Combining these two points with the above three rules (1.5) we see that the coalgebra e must be:

$$e(\sigma) \;=\; \begin{cases} \bot & \text{if } \sigma \not\to \\ (a, \sigma') & \text{if } \sigma \xrightarrow{a} \sigma' \text{ with } \sigma' \not\to \\ (a, \sigma'') & \text{if } \sigma \xrightarrow{a} \sigma' \wedge \sigma' \xrightarrow{a'} \sigma'' \end{cases}$$

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

1.2.3. Remark. The coalgebra $e\colon A^\infty \to \{\bot\} \cup (A \times A^\infty)$ illustrates the difference between states and observables. Consider an arbitrary sequence $\sigma \in A^\infty$ and write $\sigma_1 = a \cdot a_1 \cdot \sigma$ and $\sigma_2 = a \cdot a_2 \cdot \sigma$, where $a, a_1, a_2 \in A$ with $a_1 \neq a_2$. These $\sigma_1, \sigma_2 \in A^\infty$ are clearly different states of the coalgebra $e\colon A^\infty \to \{\bot\} \cup (A \times A^\infty)$, but they have the same behaviour: $\mathsf{evens}(\sigma_1) = a \cdot \mathsf{evens}(\sigma) = \mathsf{evens}(\sigma_2)$, where $\mathsf{evens} = \mathsf{beh}_e$. Such observational induistinguishability of the states σ_1, σ_2 is called bisimilarity, written as $\sigma_1 \leftrightarroweq \sigma_2$, and will be studied systematically in Chapter 3.

Oddly listed elements from a sequence

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

$$o(\sigma) \;=\; \begin{cases} \bot & \text{if } \sigma \not\to \text{ or } \sigma \xrightarrow{a} \sigma' \text{ with } \sigma' \not\to \\ (a', \sigma'') & \text{if } \sigma \xrightarrow{a} \sigma' \wedge \sigma' \xrightarrow{a'} \sigma'' \end{cases}$$

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

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

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

1.2.4. Lemma. *One has*

$$\mathit{odds} \;=\; \mathit{evens} \circ \mathit{tail},$$

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

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

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

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

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

Merging sequences

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

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

Notice the crucial reversal of arguments in the last rule.

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

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

given by:

$$m(\sigma, \tau) \;=\; \begin{cases} \bot & \text{if } \sigma \not\to \wedge\, \tau \not\to \\ (a, (\sigma, \tau')) & \text{if } \sigma \not\to \wedge\, \tau \xrightarrow{a} \tau' \\ (a, (\tau, \sigma')) & \text{if } \sigma \xrightarrow{a} \sigma'. \end{cases}$$

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

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

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

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

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

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

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

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

More coinductively defined functions for sequences can be found in [113].

Exercises

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

1.2.2. Formulate appropriate rules for the function $\mathsf{odds}\colon A^\infty \to A^\infty$ in analogy with the rules (1.5) for evens.

1.2.3. Define the empty sequence $\langle\rangle \in A^\infty$ by coinduction as a map $\langle\rangle\colon \{\bot\} \to A^\infty$.
Fix an element $a \in A$, and the define similarly the infinite sequence $\vec{a}\colon \{\bot\} \to A^\infty$ consisting only of a's.

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

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

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

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

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

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

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

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

1.3 Generality of temporal logic of coalgebras

This section will illustrate the important coalgebraic notion of invariant, and use it to introduce temporal operators like $\Box$ for henceforth, and $\Diamond$ for eventually. These operators are useful for expressing various interesting properties about states of a coalgebra. As we shall see later in Section 4.3, they can be defined for general coalgebras. But here we shall introduce them in more concrete situations—although we try to suggest the more general perspective. First, the sequences from the previous section (1.2) will be reconsidered, and next, the statements from the first section (1.1) will be used to form a rudimentary notion of class, with associated temporal operators $\Box$ and $\Diamond$ for expressing safety and liveness properties.

1.3.1 Temporal operators for sequences

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

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

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

$$\begin{aligned} \big(\bigcirc P\big)(x) &\iff \forall a \in A.\, \forall x' \in S.\, c(x) = (a, x') \Rightarrow P(x') \\ &\iff \forall a \in A.\, \forall x' \in S.\, x \xrightarrow{a} x' \Rightarrow P(x') \end{aligned} \tag{1.6}$$

In words:

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

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

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

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

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

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

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

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

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

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

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

(ii) And $\Diamond P \subseteq S$, for **eventually** P, is defined as:

$$\Diamond P \;=\; \neg\Box\neg P$$

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

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

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

The way these temporal operators $\Box$ and $\Diamond$ are defined may seem somewhat complicated at first, but will turn out to be at the right level of abstraction: as we shall see later in Section 4.3, the same formulation in terms of invariants works much more generally, for coalgebras of different types (and not just for sequence coalgebras): the definition is "generic" or "polytypic".

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

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

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

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

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

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

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

(i) Recall the termination predicate $(-) \nrightarrow$ introduced in (1.3): $x \nrightarrow$ means $c(x) = \bot$. Now consider the predicate $\Diamond((-) \nrightarrow) \subseteq S$. It holds for those states which are eventually mapped to $\bot$, *i.e.* for those states whose behaviour is a finite sequence in $A^\star \subseteq A^\infty$.

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

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

One may wish to write $a \in x$ as a more intuitive notation for $x \in \mathsf{Occ}(a)$. It means that there is a future state of x which can do an a-step, *i.e.* that a occurs somewhere in the behaviour sequent of the state x.

(iii) Now assume our set A carries an order $\leq$. Consider the predicate

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

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

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

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

Next we wish to illustrate how to reason with these temporal operators. We show that an element occurs in the merge of two sequences if and only if it occurs in at least one of the two sequences. Intuitively this is clear, but technically it is non-entirely-trivial. The proof makes essential use of invariants.

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

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

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

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

In order to derive a contradiction we form a new predicate

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

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

1. $\mathsf{merge}(\alpha, \beta) \xrightarrow{b} \mathsf{merge}(\alpha, \beta')$ because $\alpha \nrightarrow$ and $\beta \xrightarrow{b} \beta'$.
2. $\mathsf{merge}(\alpha, \beta) \xrightarrow{b} \mathsf{merge}(\beta, \alpha')$ because $\alpha \xrightarrow{b} \alpha'$.

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

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

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

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

1. If $P(\mathsf{merge}(\alpha, \beta))$ for some β, then $\mathsf{merge}(\alpha, \beta) \xrightarrow{b} \mathsf{merge}(\beta, \alpha')$, so that $\alpha' \in Q$, because $P(\mathsf{merge}(\beta, \alpha'))$, and also $b \neq a$.
2. If $P(\mathsf{merge}(\beta, \alpha))$ for some β, then there are two further cases:
   (a) If $\beta \not\rightharpoonup$, then $\mathsf{merge}(\beta, \alpha) \xrightarrow{b} \mathsf{merge}(\beta, \alpha')$, so that $\alpha' \in Q$, and $b \neq a$.
   (b) If $\beta \xrightarrow{c} \beta'$, then $\mathsf{merge}(\beta, \alpha) \xrightarrow{c} \mathsf{merge}(\alpha, \beta') \xrightarrow{b} \mathsf{merge}(\alpha', \beta')$. Thus $P(\mathsf{merge}(\alpha', \beta'))$, so that $\alpha' \in Q$, and also $b \neq a$.

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

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

1.3.2 Temporal operators for classes

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

$$\begin{array}{rcl} \mathsf{at}_1 : S & \longrightarrow & D_1 \\ & \vdots & \\ \mathsf{at}_n : S & \longrightarrow & D_n \\ \mathsf{meth}_1 : S & \longrightarrow & \{\bot\} \cup S \cup (S \times E) \\ & \vdots & \\ \mathsf{meth}_m : S & \longrightarrow & \{\bot\} \cup S \cup (S \times E) \end{array} \tag{1.7}$$

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

For such classes, like for sequences, we can define a tailor-made nexttime operator $\bigcirc$. For a predicate $P \subseteq S$, we have $\bigcirc P \subseteq S$, defined on $x \in S$ as:

$$\begin{array}{rcl} (\bigcirc P)(x) & \iff & \forall i \leq m.\;\; (\forall y \in S.\, \mathsf{meth}_i(x) = y \Rightarrow P(y)) \wedge \\ & & \qquad (\forall y \in S.\, \forall e \in E.\, \mathsf{meth}_i(x) = (y, e) \Rightarrow P(y)) \end{array}$$

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

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

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

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

Exercises

1.3.1. The nexttime operator $\bigcirc$ introduced in (1.6) is the so-called **weak** nexttime. There is an associated **strong** nexttime, given by $\neg\bigcirc\neg$. See the difference between weak and strong nexttime for sequences.

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

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

1.3.4. Prove that the finite behaviour predicate $\Diamond(- \not\rightharpoonup)$ from Example 1.3.4 (ii) is an invariant: $\Diamond(- \not\rightharpoonup) \subseteq \bigcirc\Diamond(- \not\rightharpoonup)$.
[*Hint.* For an invariant Q, consider the predicate $Q' = \neg(-) \not\rightharpoonup \cap \bigcirc Q$.]

1.3.5. Let $(A, \leq)$ be a complete lattice, *i.e.* a poset in which each subset $U \subseteq A$ has a join $\bigvee U \in A$. It is well-known that each subset $U \subseteq A$ then also has a meet $\bigwedge U \in A$, given by $\bigwedge U = \bigvee\{a \in A \mid \forall b \in U.\, a \leq b\}$.
Let $f\colon A \to A$ be a monotone function: $a \leq b$ implies $f(a) \leq f(b)$. Recall, *e.g.* from [64, Chapter 4] that such a monotone f has both a least fixed point $\mu f \in A$ and a greatest fixed point $\nu f \in A$ given by the formulas:

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

Now let $c\colon S \to \{\bot\} \cup (A \times S)$ be an arbitrary sequence coalgebra, with associated nexttime operator $\bigcirc$.
(i) Prove that $\bigcirc$ is a monotone function $\mathcal{P}(S) \to \mathcal{P}(S)$, *i.e.* that $P \subseteq Q$ implies $\bigcirc P \subseteq \bigcirc Q$, for all $P, Q \subseteq S$.
(ii) Check that $\Box P \in \mathcal{P}(S)$ is the greatest fixed point of the function $\mathcal{P}(S) \to \mathcal{P}(S)$ given by $U \mapsto P \cap \bigcirc U$.
(iii) Define for $P, Q \subseteq S$ a new predicate $P \,\mathcal{U}\, Q \subseteq S$, for "$P$ until Q" as the least fixed point of $U \mapsto Q \cup (P \cap \neg\bigcirc\neg U)$. Check that "until" is indeed a good name for $P \,\mathcal{U}\, Q$, since it can be described explicitly as:

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

[These fixed point definitions are standard in temporal logic, see *e.g.* [69, 3.24-25]. What we describe is the "strong" until. The "weak" one does not have the negations $\neg$ in its fixed point description in (iii).]

1.4 Abstractness of the coalgebraic notions

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

For this reason we need to introduce the fundamental notions of category and functor, simply because a bit of category helps enormously in presenting the theory of coalgebras, and in recognising the common structure underlying many examples. Readers who wish to learn more about categories may consider introductory texts like [17, 60, 236, 194, 31, 171], or more advanced ones such as [167, 45, 169, 130, 228].

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

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

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

1. For each pair of maps $f\colon X \to Y$ and $g\colon Y \to Z$ there is a composition map $g \circ f\colon X \to Z$. This composition operation $\circ$ is associative: if $h\colon Z \to W$, then $h \circ (g \circ f) = (h \circ g) \circ f$.

2. For each object $X \in \mathbb{C}$ there is an identity map $\mathrm{id}_X\colon X \to X$, such that id is neutral element for composition $\circ$: for $f\colon X \to Y$ one has $f \circ \mathrm{id}_X = f = \mathrm{id}_Y \circ f$. Often, the subscript X in id_X is omitted when it is clear from the context. Sometimes the object X itself is written for the identity map id_X on X.

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

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

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

(ii) Here is another degenerate example: a preorder consists of a set D with a reflexive and transitive order relation $\leq$. It corresponds to a category in which there is at most one arrow between each pair of object. Indeed, the preorder $(D, \leq)$ can be seen as a category with elements $d \in D$ as objects, and with an arrow $d_1 \to d_2$ if and only if $d_1 \leq d_2$.

(iii) Many examples of categories have certain mathematical structures as objects, and structure preserving functions between them as morphisms. Examples are:

(1) **Mon**, the category of monoids with monoid homomorphisms (preserving composition and unit).

(2) **Grp**, the category of groups with group homomorphisms (preserving composition and unit, and thereby also inverses).

(3) **PreOrd**, the category of preorders with monotone functions (preserving the order). Similarly, there is a category **PoSets** with posets as objects, and also with monotone functions as morphisms.

(4) **Dcpo**, the category of directed complete partial orders (dcpos) with continuous functions between them (preserving the order and directed joins $\bigvee$).

(5) $\mathbf{Sp}$, the category of topological spaces with continuous functions (whose inverse image preserves open subsets).

(6) **Met**, the category of metric spaces with non-expansive functions between them. Consider two objects (M_1, d_1) and (M_2, d_2) in **Met**, where $d_i\colon M_i \times M_i \to [0, \infty)$ is a distance function on the set M_i. A morphism $(M_1, d_1) \to (M_2, d_2)$ in **Met** is defined as a function $f\colon M_1 \to M_2$ between the underlying sets satisfying $d_2(f(x), f(y)) \leq d_1(x, y)$, for all $x, y \in M_1$.

(iv) An example that we shall use now and then, especially in Section 4.5 for trace semantics, is the category **REL** of sets and relations. Its objects are ordinary sets, and its morphisms $X \to Y$ are relations $R \subseteq X \times Y$. Composition of $R\colon X \to Y$ and $S\colon Y \to Z$ is given by relational composition:

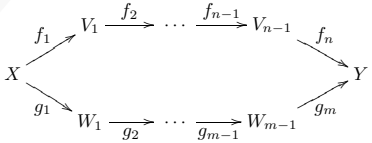
$$S \circ R \quad = \quad \{(x, z) \in X \times Z \mid \exists y \in Y.\, R(x, y) \wedge S(y, z)\} \tag{1.8}$$

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

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

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

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



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

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

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

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

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

- Given a category $\mathbb{C}$, one can form what is called the **opposite** category $\mathbb{C}^{\mathrm{op}}$ which has the same objects as $\mathbb{C}$, but the arrows reversed. Thus $f: X \to Y$ in $\mathbb{C}^{\mathrm{op}}$ if and only if $f: Y \to X$ in $\mathbb{C}$. Composition $g \circ f$ in $\mathbb{C}^{\mathrm{op}}$ is then $f \circ g$ in $\mathbb{C}$.

- Given two categories $\mathbb{C}$ and $\mathbb{D}$, we can form the **product** category $\mathbb{C} \times \mathbb{D}$. Its objects are pairs of objects (X, Y) with $X \in \mathbb{C}$ and $Y \in \mathbb{D}$. A morphism $(X, Y) \to (X', Y')$ in $\mathbb{C} \times \mathbb{D}$ consists of a pair of morphisms $X \to X'$ in $\mathbb{C}$ and $Y \to Y'$ in $\mathbb{D}$. Identities and compositions are obtained componentwise.

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

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

(i) F preserves domains and codomains: if $f: X \to Y$ in $\mathbb{C}$, then $F(f): F(X) \to F(Y)$ in $\mathbb{D}$.

(ii) F preserves identities: $F(\mathrm{id}_X) = \mathrm{id}_{F(X)}$ for each $X \in \mathbb{C}$.

(iii) F preserves composition: $F(g \circ f) = F(g) \circ F(f)$, for all maps $f: X \to Y$ and $g: Y \to Z$ in $\mathbb{C}$.

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

Further, given two functors $F: \mathbb{C} \to \mathbb{D}$ and $G: \mathbb{D} \to \mathbb{E}$, there is a composite functor $G \circ F: \mathbb{C} \to \mathbb{E}$. It is given by $X \mapsto G(F(X))$ and $f \mapsto G(F(f))$.

Often we are a bit sloppy in the use of parentheses in functor applications. Thus FX and Ff are $F(X)$ and $F(f)$. Similarly, GFX is $G(F(X))$.

1.4.4. Examples. (i) Consider two monoids $(M, +, 0)$ and $(N, \cdot, 1)$ as categories, like in Example 1.4.2 (i). A functor $f: M \to N$ is then the same as a monoid homomorphism: it preserves the composition operation and unit element.

(ii) Similarly, consider two preorders $(D, \leq)$ and $(E, \sqsubseteq)$ as categories, like in Example 1.4.2 (ii). A functor $f: D \to E$ is then nothing but a monotone function: $x \leq y$ implies $f(x) \sqsubseteq f(y)$.

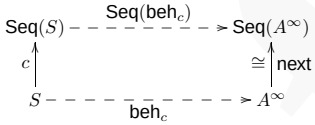
(iii) Frequently occurring examples of functors are so-called **forgetful** functors. They forget part of the structure of their domain. For instance, there is a forgetful functor $\mathbf{Mon} \to \mathbf{Sets}$ mapping a monoid $(M, +, 0)$ to its underlying set M, and mapping a monoid homomorphism f to f, considered as a function between sets. Similarly, there is a forgetful functor $\mathbf{Grp} \to \mathbf{Mon}$ mapping groups to monoids by forgetting their inverse operation.

(iv) There is a "graph" functor $\mathbf{Sets} \to \mathbf{REL}$. It maps a set X to X itself, and a function $f: X \to Y$ to the corresponding graph relation $\mathrm{Graph}(f) \subseteq X \times Y$ given by $\mathrm{Graph}(f) = \{(x, y) \mid f(x) = y\}$.

(v) Recall from Section 1.2 that sequence coalgebras were described as functions of the form $c: S \to \{\bot\} \cup (A \times S)$. Their codomain can be described via a functor $\mathsf{Seq}: \mathbf{Sets} \to \mathbf{Sets}$. It maps a set X to the set $\{\bot\} \cup (A \times X)$. And it sends a function $f: X \to Y$ to a function $\{\bot\} \cup (A \times X) \to \{\bot\} \cup (A \times Y)$ given by:

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

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



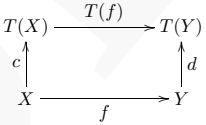
In this book we shall be especially interested in **endofunctors**, *i.e.* in functors $\mathbb{C} \to \mathbb{C}$ from a category $\mathbb{C}$ to itself. In many cases this category will $\mathbb{C}$ will simply be $\mathbf{Sets}$, the category of sets and functions. Often we say that a mapping $A \mapsto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

We can now introduce coalgebras in full generality.

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

(i) An F-**coalgebra**, or just a **coalgebra** when F is understood, consists of an object $X \in \mathbb{C}$ together with a morphism $c: X \to F(X)$. As before, we often call X the state space, and c the transition or coalgebra structure.

(ii) A **homomorphism of coalgebras**, or a **map of coalgebras**, or a **coalgebra map**, from one coalgebra $c: X \to T(X)$ to another coalgebra $d: Y \to T(Y)$ consists of a morphism $f: X \to Y$ in $\mathbb{C}$ which commutes with the structures, in the sense that the following diagram commutes.



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

The abstractness of the notion of coalgebra lies in the fact that it can be expressed in any category. So we need not only talk about coalgebras in $\mathbf{Sets}$, as we have done so far, but we can also consider coalgebras in other catgories. For instance, one can have coalgebras in **PreOrd**, the category of preorders. In that case, the state space is a preorder, and the coalgebra structure is a monotone function. Similarly, a coalgebra in the category **Mon** of monoids has a monoid as state space, and a structure which preserves this monoid structure. We can even have a coalgebra in a category $\mathbf{CoAlg}(F)$ of coalgebras. We briefly mention some examples.

- Real numbers (and also Baire and Cantor space) are described in [193, Theorem 5.1] as final coalgebras (via continued fractions, see also [187]) of an endofunctor on the category $\mathbf{PoSets}$.

- So-called descriptive general frames (special models of modal logic) appear in [162] as coalgebras of the Vietoris functor on the category of Stone spaces.
- At several places in this book we shall see coalgebra of endofunctors other than sets. For instance, Exercise 1.4.5 mentions invariants as coalgebras of endofunctors on poset categories, and Example 2.3.10 and Exercise 2.3.5 describe streams with their topology as final coalgebra in the category of topological spaces. Section 4.5 introduces traces of suitable coalgebras via coalgebra homomorphism to a final coalgebra in the category **REL** of sets with relations as morphisms.

In the next few chapters we shall concentrate on coalgebras in $\mathbf{Sets}$, but occasionally this more abstract perspective will be useful.

Exercises

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

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

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

(i) Check that $A^\star$ carries a monoid structure given by concatenation of sequences, with the empty sequence $\langle\rangle$ as neutral element.

(ii) Check that the assignment $A \mapsto A^\star$ yields a functor $\mathbf{Sets} \to \mathbf{Mon}$ by mapping a function $f\colon A \to B$ between sets to the function $f^\star\colon A^\star \to B^\star$ given by $\langle a_0, \ldots, a_n\rangle \mapsto \langle f(a_0), \ldots, f(a_n)\rangle$.
[Be aware of what needs to be checked: $f^\star$ must be a monoid homomorphism, and $(-)^\star$ must preserve composition of functions and identity functions.]

(iii) Prove that $A^\star$ is the **free monoid on** A: there is the singleton-sequence insertion map $\eta\colon A \to A^\star$ which is universal among all mappings of A into a monoid: for each monoid $(M, 0, +)$ and function $f\colon A \to M$ there is a unique monoid homomorphism $g\colon A^\star \to M$ with $g \circ \eta = f$.

1.4.4. Reconsider from Section 1.1 the statements with exceptions of the form $S \to \{\bot\} \cup S \cup (S \times E)$.

(i) Prove that the assignment $X \mapsto \{\bot\} \cup X \cup (X \times E)$ is functorial, so that statements are coalgebras for this functor.

(ii) Show that all the operations $\mathsf{at}_1, \ldots, \mathsf{at}_n, \mathsf{meth}_1, \ldots, \mathsf{meth}_m$ of a class as in (1.7) can also be described as a single coalgebra, namely of the functor:

$$X \longmapsto D_1 \times \cdots D_n \times \underbrace{(\{\bot\} \cup X \cup (X \times E)) \times \cdots \times (\{\bot\} \cup X \cup (X \times E))}_{m \text{ times}}$$

1.4.5. Recall the nexttime operator $\bigcirc$ for a sequence coalgebra $c\colon S \to \mathsf{Seq}(S) = \{\bot\} \cup (A \times S)$ from the previous section. Check that it forms a monotone function $\mathcal{P}(S) \to \mathcal{P}(S)$—with respect to the inclusion order—and thus a functor. Conclude that invariants are $\bigcirc$-coalgebras!

Chapter 2

Preliminaries on coalgebras and algebras

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

2.1 Constructions on sets

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

These basic constructions on sets are instances of more general constructions in categories. We shall give a perspective on these categorical formulations, but we do not overemphasise this point. Readers without much familiarity with the theory of categories may then still follow the development, and readers who are quite comfortable with categories will recognise this wider perspective anyway.

Products

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

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

There are then obvious projection functions $\pi_1\colon X \times Y \to X$ and $\pi_2\colon X \times Y \to Y$ by $\pi_1(x, y) = x$ and $\pi_2(x, y) = y$. Also, for functions $f\colon Z \to X$ and $g\colon Z \to Y$ there is a

tuple (or pairing) function $\langle f, g\rangle \colon Z \to X \times Y$ given by $\langle f, g\rangle(z) = (f(z), g(z)) \in X \times Y$ for $z \in Z$. Here are some basic equations which are useful in computations.

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

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

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

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

Interestingly, the product operation $(X, Y) \mapsto X \times Y$ 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 $f \times g$ namely:

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

Notice that the symbol $\times$ is overloaded: it is used both on sets and on functions. This product function $f \times g$ can also be described in terms of projections and pairing as $f \times g = \langle f \circ \pi_1, g \circ \pi_2\rangle$. It is easily verified that the operation $\times$ on functions satisfies

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

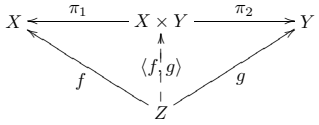
This expresses that the product $\times$ is *functorial*: it does not only apply to sets, but also to functions; and it does so in such a way that identity maps and compositions are preserved. The product operation $\times$ is a functor $\mathbf{Sets} \times \mathbf{Sets} \to \mathbf{Sets}$, from the product category $\mathbf{Sets} \times \mathbf{Sets}$ of $\mathbf{Sets}$ with itself, to $\mathbf{Sets}$.

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

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

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

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



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

What we have described is the product $X \times Y$ of *two* sets / objects X, Y. For a given X, we shall write $X^n = X \times \cdots \times X$ for the n-fold product. The definition of product can easily be generalised to products $\prod_{i\in I} X_i$ of collections $(X_i)_{i\in I}$ indexed by a set I. Thus, the n-fold product X^n is the product $\prod_{1\leq i\leq n} X$ of n-copies of X.

We shall be particularly interested in the *empty* product, over the empty index set $I = \emptyset$.

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

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

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

Coproducts

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

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

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

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

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

$$\begin{array}{rcl} [f, g] \circ \kappa_1 & = & f \\ [f, g] \circ \kappa_2 & = & g \\ [\kappa_1, \kappa_2] & = & \mathrm{id}_{X+Y} \\ h \circ [f, g] & = & [h \circ f, h \circ g]. \end{array} \tag{2.4}$$

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

$$\frac{X \longrightarrow Z \qquad Y \longrightarrow Z}{X + Y \longrightarrow Z} \tag{2.5}$$

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

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

$$(f+g)(w) \quad = \quad \begin{cases} (f(x), 1) & \text{if } w = (x, 1) \\ (g(y), 2) & \text{if } w = (y, 2). \end{cases} \tag{2.6}$$

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

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

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

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

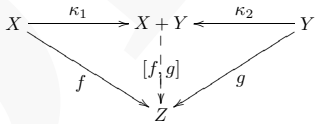
We should emphasise that a coproduct $+$ is very different from ordinary union $\cup$. For example, $\cup$ is idempotent: $X \cup X = X$, but there is not even an isomorphism between $X + X$ and X (if $X \neq \emptyset$). Union is an operation on subsets, whereas coproduct is an operation on sets.

Also the coproduct $+$ in $\mathbf{Sets}$ is an instance of a more general categorical notion of coproduct.

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

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

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



Also, there is a notion of empty coproduct.

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

Like for products, one says that a category has finite coproducts when it has binary coproducts $+$ plus an initial object. In that case one can form coproducts $X_1 + \cdots + X_n$ for any finite list of objects X_i.

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

Whereas products are very familiar, coproducts are relatively unknown. From a purely categorical perspective, they are not more difficult than products, because they are their

duals (see Exercise 2.1.3 below). But in a non-categorical setting the cotuple $[f, g]$ is a bit complicated, because it involves variable binding: in a term calculus one can write $[f, g](z)$ for instance as:

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

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

Another reason why coproducts are not so standard in mathematics is probably that in many algebraic structures coproducts coincide with products (*e.g.* for monoids or vector spaces, see Exercise 2.1.6), and in many continuous structures they do not exist (like in categories of domains).

However, within the theory of coalgebras coproducts play an important role. They occur in many functors F used to describe coalgebras (namely as F-coalgebras, see Definition 1.4.5), in order to capture different output options, like normal and abnormal termination in Section 1.1. Thus it is important within this setting to take coproducts seriously.

Exponents

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

$$\begin{array}{rcl} \mathrm{ev} \circ \Lambda(f) \times \mathrm{id}_X & = & f \\ \Lambda(\mathrm{ev}) & = & \mathrm{id}_{Y^X} \\ \Lambda(f) \circ h & = & \Lambda(f \circ h \times \mathrm{id}_X). \end{array} \tag{2.7}$$

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

$$\frac{Z \times X \longrightarrow Y}{Z \longrightarrow Y^X} \tag{2.8}$$

We have seen that both the product $\times$ and the coproduct $+$ give rise to functors $\mathbf{Sets} \times \mathbf{Sets} \to \mathbf{Sets}$. The situation for exponents is more subtle, because of the so-called contravariance in the first argument. This leads to an exponent functor $\mathbf{Sets}^{\mathrm{op}} \times \mathbf{Sets} \to \mathbf{Sets}$, involving an opposite category for its first argument. We will show how this works.

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

$$h^k(f) \quad = \quad h \circ f \circ k. \tag{2.9}$$

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

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

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

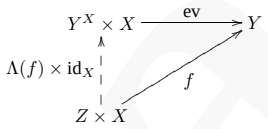
$$\begin{array}{rcl}\big(h_2^{k_2} \circ h_1^{k_1}\big)(f) &=& \big(h_2^{k_2}\big(h_1 \circ f \circ k_1\big)\\ &=& h_2 \circ h_1 \circ f \circ k_1 \circ k_2\\ &=& \big((h_2 \circ h_1)^{(k_2 \circ k_1)}\big)(f).\end{array}$$

We conclude this discussion of exponents with the categorical formulation.

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

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

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



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

Powersets

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

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

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

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

Reversal and composition are two basic constructions on relations. For a relation $R \subseteq X \times Y$ we shall write $R^{-1} \subseteq Y \times X$ for the reverse relation given by $yR^{-1}x$ iff xRy. If we have another relation $S \subseteq Y \times Z$ we can describe the composition of relations $S \circ R$ as a new relation $(S \circ R) \subseteq X \times Z$, via: $x(S \circ R)z$ iff $\exists y \in Y.\, R(x, y) \wedge S(y, z)$, as already described in (1.8).

Often we are interested in relations $R \subseteq X \times X$ on a single set X. Of special interest then is the equality relation $\mathrm{Eq}(X) \subseteq X \times X$ given by $\mathrm{Eq}(X) = \{(x, y) \in X \times X \mid x = y\} = \{(x, x) \mid x \in X\}$. As we saw in Example 1.4.2 (iv), sets and relations form a category **REL**.

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

$$\begin{array}{rcl}\mathcal{P}(f)(U) &=& \{f(x) \mid x \in U\}\\ &=& \{y \in Y \mid \exists x \in X.\, f(x) = y \wedge x \in U\}.\end{array} \tag{2.10}$$

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

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

$$f^{-1}(U) \;=\; \{x \mid f(x) \in U\}. \tag{2.11}$$

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

$$\frac{\coprod_f(U) \;\subseteq\; V}{U \;\subseteq\; f^{-1}(V)} \tag{2.12}$$

See also in Exercise 2.1.12 below.

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

$$\frac{X \longrightarrow \mathcal{P}(Y)}{\text{relations} \subseteq Y \times X} \tag{2.13}$$

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

This formalisation of this special property in categorical language yields so-called power objects. The presence of such objects is a key feature of "toposes". The latter are categorical set-like universes, with constructive logic. They form a topic that goes beyond the introductory material covered in this text. The interested reader is referred to the extensive literature on toposes [149, 94, 30, 169, 45].

Finally, we shall often need the *finite* powerset $\mathcal{P}_{\mathrm{fin}}(X) = \{U \in \mathcal{P}(X) \mid U \text{ is finite}\}$.

Exercises

2.1.1. Verify in detail the bijective correspondences (2.2), (2.5), (2.8) and (2.13).

2.1.2. Consider a poset $(D, \leq)$ as a category. Check that the product of two elements $d, e \in D$, if it exists, is the meet $d \wedge e$. And a coproduct of d, e, if it exists, is the join $d \vee e$.
Similarly, show that a final object is a top element $\top$ (with $d \leq \top$, for all $d \in D$), and that an initial object is a bottom element $\bot$ (with $\bot \leq d$, for all $d \in D$).

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

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

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

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

(i) Check that the singleton monoid 1 is both an initial and a final object in **Mon**.

(ii) Given two monoids $(M_1, +_1, 0_1)$ and $(M_2, +_2, 0_2)$, define a product monoid $M_1 \times M_2$ with componentwise addition $(x, y) + (x', y') = (x +_1 x', y +_2 y')$ and unit $(0_1, 0_2)$. Prove that $M_1 \times M_2$ is again a monoid, which forms a product in the category **Mon** with the standard projection maps $M_1 \xleftarrow{\pi_1} M_1 \times M_2 \xrightarrow{\pi_2} M_2$.

(iii) Note that there are also coprojections $M_1 \xrightarrow{\kappa_1} M_1 \times M_2 \xleftarrow{\kappa_2} M_2$, given by $\kappa_1(x) = (x, 0_2)$ and $\kappa_2(y) = (0_1, y)$ which are monoid homomorphisms, and which make $M_1 \times M_2$ at the same time the coproduct of M_1 and M_2 in **Mon**.

[*Hint.* Define the cotuple $[f, g]$ as $x \mapsto f(x) + g(x)$.]

2.1.7. Show that in $\mathbf{Sets}$ products distribute over coproducts, in the sense that the canonical maps

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

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

are isomorphisms. Categories in which this is the case are called **distributive**, see [103] for an investigation of such distributivities in categories of coalgebras.

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

$$X \times Y \cong Y \times X \qquad (X \times Y) \times Z \cong X \times (Y \times Z) \qquad 1 \times X \cong X$$

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

$$X + Y \cong Y + X \qquad (X + Y) + Z \cong X + (Y + Z) \qquad 0 + X \cong X$$

[This means that both the finite product and coproduct structure in a category yields so-called *symmetric monoidal* structure. See [167, 45] for more information.]

(iii) Next, assume that our category also has exponents. Prove that:

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

And also that:

$$Z^{X+Y} \cong Z^X \times Z^Y \qquad Z^{X \times Y} \cong \left(Z^Y\right)^X \qquad (X \times Y)^Z \cong X^Z \times Y^Z$$

2.1.9. Check that:

$$\mathcal{P}(0) \cong 1 \qquad \mathcal{P}(1) \cong 2 = \{0, 1\} \qquad \mathcal{P}(X + Y) \cong \mathcal{P}(X) \times \mathcal{P}(Y).$$

And similarly for the finite powerset $\mathcal{P}_{\mathrm{fin}}(-)$ instead of $\mathcal{P}(-)$.

2.1.10. Show that the finite powerset also forms a functor $\mathcal{P}_{\mathrm{fin}}(-) \colon \mathbf{Sets} \to \mathbf{Sets}$.

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

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

(i) the direct image $\coprod_f \colon \mathcal{P}(X) \to \mathcal{P}(Y)$ preserves all joins, and that the inverse image $f^{-1}(-) \colon \mathcal{P}(Y) \to \mathcal{P}(X)$ preserves not only joins but also meets and negation (*i.e.* all the Boolean structure);

(ii) there is a Galois connection $\coprod_f(U) \subseteq V \iff U \subseteq f^{-1}(V)$, as claimed in (2.12);

(iii) there is a product function $\prod_f \colon \mathcal{P}(X) \to \mathcal{P}(Y)$ given by $\prod_f(U) = \{y \in Y \mid \forall x \in X.\, f(x) = y \Rightarrow x \in U\}$, with a Galois connection $f^{-1}(V) \subseteq U \iff V \subseteq \prod_f(U)$.

2.1.13. Let $\mathbb{C}$ be a category with finite coproducts $(0, +)$, and let F be an arbitrary endofunctor $\mathbb{C} \to \mathbb{C}$. Prove that the category $\mathbf{CoAlg}(F)$ of F-coalgebras then also has finite coproducts, by showing that:

(i) the unique map $0 \to F(0)$ is the initial coalgebra;

(ii) the cotuple $[F(\kappa_1) \circ c, F(\kappa_2) \circ d] \colon X + Y \to F(X + Y)$ of two coalgebras $c \colon X \to F(X)$ and $d \colon Y \to F(Y)$ is their coproduct in $\mathbf{CoAlg}(F)$.

Generalise this to arbitrary (set-indexed) coproducts $\coprod_{i \in I} X_i$.

2.1.14. For two parallel functions $f, g \colon X \to Y$ between sets X, Y one can form their **coequaliser** $q \colon Y \to Q$ in a diagram,

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

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

(i) Check that Q can be defined as the quotient Y/R, where $R \subseteq Y \times Y$ is the least equivalence relation containing all pairs $(f(x), g(x))$ for $x \in X$.

(ii) Prove that for an arbitrary functor $F \colon \mathbf{Sets} \to \mathbf{Sets}$ the associated category of coalgebras $\mathbf{CoAlg}(F)$ also has coequalisers: for two parallel homomorphisms $f, g \colon X \to Y$ between coalgebras $c \colon X \to F(X)$ and $d \colon Y \to F(Y)$ there is by universality an induced coalgebra structure $Q \to F(Q)$ on the coequaliser Q of the underlying function, yielding a diagram of coalgebras

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

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

2.2 Polynomial functors and their coalgebras

Earlier in Definition 1.4.5 we have seen the general notion of a coalgebra as a map $X \to F(X)$ in a category $\mathbb{C}$, where F is a functor $\mathbb{C} \to \mathbb{C}$. Here, in this section and in much of the rest of this text we shall concentrate on a more restricted situation: as category $\mathbb{C}$ we use the category $\mathbf{Sets}$ of ordinary sets and functions. And as functors $F \colon \mathbf{Sets} \to \mathbf{Sets}$ we shall use so-called polynomial functors, like $F(X) = A + (B \times X)^C$. These are functors built up inductively from certain simple basic functors, using products, coproducts, exponents and powersets for forming new functors. There are three reasons for this restriction to polynomial functors.

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

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

2.2.1. Definition. (i) The collection SPF of **simple polynomial functors** is the least class of functors $\mathbf{Sets} \to \mathbf{Sets}$ satisfying the following clauses.

(1) The identity functor $\mathbf{Sets} \to \mathbf{Sets}$ is in SPF.

(2) For each set A, the constant functor $A \colon \mathbf{Sets} \to \mathbf{Sets}$ is in SPF. Recall that it maps every set X to A, and every function f to the identity id_A on A.

(3) If both F and G are in SPF, then so is the product functor $F \times G$, defined as $X \mapsto F(X) \times G(X)$. On functions it is defined as $f \mapsto F(f) \times G(f)$, see (2.3).

(4) If F and G are in SPF, then also their coproduct $F+G$ given by $X \mapsto F(X)+G(X)$ is in SPF. A function f is sent by this coproduct functor to $F(f) + G(f)$, see (2.6).

(5) For each set A, if F in SPF, then so is the "constant" exponent F^A defined as $X \mapsto F(X)^A$. It sends a function $f\colon X \to Y$ to the function $F(f)^A = F(f)^{\mathrm{id}_A}$ which maps $h\colon A \to F(X)$ to $F(f) \circ h\colon A \to F(Y)$, see (2.9).

(ii) The class KPF of **(Kripke) polynomial functors** is the superset of SPF defined by the above clauses (1)–(5), with 'SPF' replaced by 'KPF', plus two additional rules:

(6) If F is in KPF, then so is the powerset $\mathcal{P}(F)$, defined as $X \mapsto \mathcal{P}(F(X))$ on sets, and as $f \mapsto \mathcal{P}(F(f))$ on functions, see (2.10).

(7) If $F \in \mathsf{KPF}$, then also $F^\star \in \mathsf{KPF}$, where the functor $F^\star$ maps a set X to the set $F(X)^\star$ of finite sequences of elements of $F(X)$. On a function f it is defined as $F(f)^\star$, like in Exercise 1.4.3 (ii).

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

Thus, when we write 'polynomial functor' we mean 'Kripke polynomial functor'. We shall occasionally restrict ourselves to 'simple' polynomial functors (without powersets), like in Chapter 5 when we introduce specifications for coalgebras. The above clauses yield a reasonable collection of functors to start from, but we could of course have included some more constructions in our definition of polynomial functor—like iterations via initial and final (co)algebras, see *e.g.* [112, 208, 147], as used in the experimental programming language Charity [55, 53, 56]. There are thus interesting functors which are out of the "polynomial scope", see for instance probability distribution functors in Exercise 2.2.9, or "dependent" polynomial functors in Exercise 2.2.4. However, the above clauses suffice for many examples in this introduction.

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

2.2.1 Statements and sequences

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

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

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

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

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

And so we recognise these statements as coalgebras

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

of the simple polynomial functors:

$$\begin{array}{rcl} F &=& 1 + \mathrm{id} \\ &=& \Big(X \longmapsto 1 + X\Big) \end{array} \quad \text{and} \quad \begin{array}{rcl} G &=& 1 + \mathrm{id} + (\mathrm{id} \times E) \\ &=& \Big(X \longmapsto 1 + X + (X \times E)\Big). \end{array}$$

Thus, these functors determine the kind of computations.

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

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

i.e. as coalgebras:

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

of the simple polynomial functor $1 + (A \times \mathrm{id})$. This functor was called Seq in Example 1.4.4 (v). Again, the functor determines the kind of computations: either fail, or produce an element in A together with a successor state. We could change this a bit and drop the fail-option. In that case, each state yields an element in A with a successor state. This different kind of computation is captured by a different polynomial functor, namely by $A \times \mathrm{id}$. A coalgebra of this functor is a function

$$S \longrightarrow A \times S$$

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

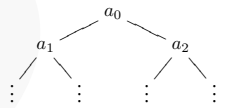
2.2.2 Trees

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

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

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

of the simple polynomial functor $A \times \mathrm{id} \times \mathrm{id}$. Indeed, given a state $x \in S$, a one-step computation yields a triple (a_0, x_1, x_2) of an element $a_0 \in A$ and two successor states $x_1, x_2 \in S$. Continuing the computation with both x_1 and x_2 yields two more elements in A, and four successor states, *etc.* This yields for each $x \in S$ an infinite binary tree with one label from A at each node:



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

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

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

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

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

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

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

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

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

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

The resulting behaviour trees will have elements from A at their leaves.

2.2.3 Deterministic automata

Automata are very basic structures in computing, used in various areas: language theory, text processing, complexity theory, parallel and distributed computing, circuit theory, *etc.* Their state-based, dynamical nature makes them canonical examples of coalgebras. There are many versions of automata, but here we shall concentrate on the two most basic ones: deterministic and non-deterministic. For more information on the extensive theory of automata, see for example [14, 67, 221].

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

First, we shall massage these ingredients into coalgebraic shape. Via the bijective correspondence (2.8) for exponents, the transition funcion $S \times A \to S$ can also be understood as a map $S \to S^A$. And the subset $F \subseteq S$ of final states corresponds to a characteristic function $S \to \{0, 1\}$. These two functions $S \to S^A$ and $S \to \{0, 1\}$ can be combined to a single function,

$$S \longrightarrow S^A \times \{0, 1\}$$

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

Here we shall stretch this notion a little bit, and replace the set $\{0, 1\}$ by an arbitary set B of "observable" outputs. Thus, what shall call a **deterministic automaton** is a coalgebra

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

of the simple polynomial functor $\mathrm{id}^A \times B$. Such automata are sometimes called Mealy machines.

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

Often one considers automata with a *finite* state space. This is not natural in a coalgebraic setting, because the state space is considered to be a black box, about which essentially nothing is known—except what can be observed via the operations. Hence we shall work with arbitrary state spaces, without assuming finiteness.

Assume we have a state $x \in S$ of such a coalgebra / deterministic automaton $\langle \delta, \epsilon \rangle\colon S \to S^A \times B$. Applying the output function $\epsilon\colon S \to B$ to x yields an immediate observation $\epsilon(x) \in B$. For each element $a_1 \in A$ we can produce a successor state $\delta(x)(a_1) \in S$; it also gives rise to an immediate observation $\epsilon(\delta(x)(a_1)) \in B$, and for each $a_2 \in A$ a successor state $\delta(\delta(x)(a_1))(a_2) \in S$, *etc.* Thus, for each finite sequence $\langle a_1, \ldots, a_n \rangle \in A^\star$ we can observe an element $\epsilon(\delta(\cdots\delta(x)(a_1)\cdots)(a_n)) \in B$. Everthing we can possibly observe about the state x is obtained in this way, namely as a function $A^\star \to B$. Such behaviours will form the states of the final coalgebra, see Proposition 2.3.5 in the next section.

For future reference we shall be a bit more precise about what we have just said. The behaviours can best be described via an iterated transition function

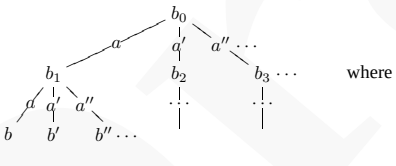
$$S \times A^\star \xrightarrow{\delta^*} S \qquad \text{defined as} \qquad \begin{cases} \delta^*(x, \langle\rangle) &= x \\ \delta^*(x, a \cdot \sigma) &= \delta^*(\delta(x)(a), \sigma) \end{cases}$$

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

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

$$\mathsf{beh}(x) \stackrel{\text{def}}{=} \epsilon(\delta^*(x, \sigma)) \in B \tag{2.14}$$

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



where

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

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

Here is a special way to obtain deterministic automata. A standard result (see *e.g.* [118, 8.7]) in the theory of differential equations says that unique solutions to such equations give rise to monoid actions, see Exercises 1.4.1 and 2.2.6. Such a monoid action may be of the form $X \times \mathbb{R}_{\geq 0} \to X$, where X is the set of states and $\mathbb{R}_{\geq 0}$ is the set of non-negative reals with monoid structure $(+, 0)$ describing the input (which may be understood as time). In this context monoid actions are sometimes called *flows*, motions, solutions or trajectories, see Exercise 2.2.8 for an example.

Exercise 2.2.10 below contains an account of linear dynamical systems which is very similar to the above approach to deterministic automata. It is based on the categorical analysis by Arbib and Manes [15, 16, 19, 18, 20] of Kalman's [153] module-theoretic approach to linear systems.

In Lemma 2.2.2 there is a description of what coalgebra homomorphisms mean for automata.

2.2.4 Non-deterministic automata and transition systems

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

$$\dfrac{\dfrac{\dfrac{S \longrightarrow \mathcal{P}(S)^A}{S \times A \longrightarrow \mathcal{P}(S)}\,(2.8)}{\text{relations} \subseteq (S \times A) \times S}\,(2.13)}{\dfrac{\text{relations} \subseteq S \times (A \times S)}{S \longrightarrow \mathcal{P}(A \times S)}\,(2.13)}$$

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

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

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

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

of a Kripke polynomial functor $\mathcal{P}(\mathrm{id})^A \times B$, where A is the set of its inputs, and B the set of its outputs or observations. Like before, the coalgebra consists of a **transition function** $\delta\colon S \to \mathcal{P}(S)^A$, and an **observation function** $\epsilon\colon S \to B$. For such a non-deterministic automaton we shall use the same notation as for deterministic ones: $x \xrightarrow{a} x'$ stands for $x' \in \delta(x)(a)$, and $x \downarrow b$ means $\epsilon(x) = b$. New notation is $x \overset{a}{\not\to}$, which means $\delta(x)(a) = \emptyset$, *i.e.* there is no successor state x' such that x can do an a-step $x \xrightarrow{a} x'$ to x'.

In general, for a given x and a there may be multiple (many or no) x' with $x \xrightarrow{a} x'$, but there is precisely one b with $x \downarrow b$. We also use the combined notation $(x \downarrow b) \xrightarrow{a} (x' \downarrow b')$ for: $x \downarrow b$ and $x \xrightarrow{a} x'$ and $x' \downarrow b'$.

Like for deterministic automata we wish to define the behaviour of a state from transitions $(x \downarrow b) \xrightarrow{a} (x' \downarrow b')$ by omitting the states and keeping the inputs and outputs. But in the non-deterministic case things are not so easy because there may be multiple successor states. More technically, there are no final coalgebras for functors $\mathcal{P}(\mathrm{id})^A \times B$ describing non-deterministic automata, see Proposition 2.3.8 and the discussion in the preceding paragraph. However, for non-deterministic automata one can consider other forms of behaviour, such as traces, see Section 4.5.

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

- An **unlabeled transition system** (UTS) is a non-deterministic automaton with trivial inputs and outputs: $A = 1$ and $B = 1$. It is thus nothing else but a relation $\to \subseteq S \times S$ on a set of states. UTSs are very basic dynamical systems, but they are important for instance as a basis for model checking: automatic state exploration for proving or disproving properties about systems, see for instance [175, 69].

- A **labeled transition system**, (LTS) introduced in [199], is a non-deterministic automaton with trivial output: $B = 1$. It can be identified with a relation $\to \subseteq S \times A \times S$. Labeled transition systems play an important rôle in the theory of processes, see *e.g.* [38].

- A **Kripke structure** is a non-deterministic automaton $S \longrightarrow \mathcal{P}(S) \times \mathcal{P}(\mathsf{AtProp})$ with trivial input ($A = 1$), and special output: $B = \mathcal{P}(\mathsf{AtProp})$, for a set AtProp of atomic propositions. The transition function $\delta\colon S \to \mathcal{P}(S)$ thus corresponds to an unlabeled transition system. And the observation function function $\epsilon\colon S \to \mathcal{P}(\mathsf{AtProp})$ tells for each state which of the atomic propositions are true (in that state). Such a function, when written as $\mathsf{AtProp} \to \mathcal{P}(S)$ is often called a valuation. Kripke structures are fundamental in the semantics of modal logic, see *e.g.* [43] or [70]. The latter reference describes Kripke structures for "multiple agents", that is, as coalgebras $S \longrightarrow \mathcal{P}(S) \times \cdots \times \mathcal{P}(S) \times \mathcal{P}(\mathsf{AtProp})$ with multiple transition functions.

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

2.2.2. Lemma. (i) *Consider two deterministic automata $X \to X^A \times B$ and $Y \to Y^A \times B$ as coalgebras. A function $f\colon X \to Y$ is then a homomorphism of coalgebras if and only if*

(1) $x \downarrow b \Longrightarrow f(x) \downarrow b$;

(2) $x \xrightarrow{a} x' \Longrightarrow f(x) \xrightarrow{a} f(x')$.

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

(1) $x \downarrow b \Longrightarrow f(x) \downarrow b$;

(2) $x \xrightarrow{a} x' \Longrightarrow f(x) \xrightarrow{a} f(x')$;

(3) $f(x) \xrightarrow{a} y \Longrightarrow \exists x' \in S_1.\, f(x') = y \wedge x \xrightarrow{a} x'$.

Such a function f is sometimes called a zig-zag morphism, see [36]. ❑

Note that the analogue of point (3) in (ii) trivially holds for deterministic automata.

2.2.5 Context-free grammars

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

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

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

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

A simple example is the grammar with $V = \{v\}$, $A = \{a, b\}$ and $g(v) = \{\langle\rangle, avb\}$. It thus involves two productions $v \longrightarrow \langle\rangle$ and $v \longrightarrow a \cdot v \cdot b$. This grammar generates the

language of words $a^n b^n$ consisting of a number of a's followed by an equal number of b's. Such a grammar is often written in "Backus Naur Form" (BNF) as $v ::= \langle\rangle \mid avb$.

A derivation of a word imposes a structure on the word that is generated. This structure may be recognised in an arbitrary word in a process called **parsing**. This is one of the very first things a compiler does (after lexical analysis), see for instance [13]. See Example 4.5.2 for the parsed language associated with a CFG via a trace semantics defined by coinduction.

2.2.6 Non-well-founded sets

Non-well-founded sets form a source of examples of coalgebras which has been of historical importance in the development of the area. Recall that in ordinary set theory there is a foundation axiom (see *e.g.* [77, Chapter II, §5]) stating that there are no infinite descending $\in$-chains $\cdots \in x_2 \in x_1 \in x_0$. This foundation axiom is replaced by an anti-foundation axiom in [76], and also in [4], allowing for non-well-founded sets. The second reference [4] received much attention; it formulated an anti-foundation axiom as: every graph has a unique decoration. This can be reformulated easily in coalgebraic terms, stating that the universe of non-well-founded sets is a final coalgebra for the (special) powerset functor $\wp$ on the category of classes and functions:

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

Incidentally, the "initial algebra" (see Section 2.4) of this functor is the ordinary universe of well-founded sets, see [233, 231] for details.

Aczel developed his non-well-founded set theory in order to provide a semantics for Milner's theory CCS [179] of concurrent processes. An important contribution of this work is the link it established between the proof principle in process theory based on bisimulations (going back to [179, 189]), and coinduction as proof principle in the theory of coalgebras—which will be described here in Section 3.4. This work formed a source of much inspiration in the semantics of programming languages [233] and also in logic [35].

Exercises

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

2.2.2. Describe the kind of trees that can arise as behaviours of coalgebras:
(i) $S \longrightarrow A + (A \times S)$
(ii) $S \longrightarrow A + (A \times S) + (A \times S \times S)$

2.2.3. Check, using Exercise 2.1.9, that non-deterministic automata $X \to \mathcal{P}(X)^A \times 2$ can equivalently be described as transition systems $X \to \mathcal{P}(1 + (A \times X))$. Work out the correspondence in detail.

2.2.4. Consider an indexed collection $(A_i)_{i \in I}$, and define the associated "dependent" polynomial functor $\mathbf{Sets} \to \mathbf{Sets}$ by

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

(i) Prove that we get a functor in this way.
(ii) Check that all simple polynomial functors are dependent—by finding suitable collections $(A_i)_{i \in I}$ for each of them.

[These functors are studied in the context of so-called W-types in dependent type theory for well-founded trees, see for instance [1, 188].]

2.2.5. (i) Notice that the behaviour function $\mathsf{beh} \colon S \to B^{A^\star}$ from (2.14) for a deterministic automaton satisfies:

$$\begin{array}{rcl} \mathsf{beh}(x)(\langle\rangle) & = & \epsilon(x) \\ & = & b, \quad \text{where } x \downarrow b \\ \mathsf{beh}(x)(a \cdot \sigma) & = & \mathsf{beh}(\delta(x)(a))(\sigma) \\ & = & \mathsf{beh}(x')(\sigma), \quad \text{where } x \xrightarrow{a} x'. \end{array}$$

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

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

2.2.6. Check that the iterated transition function $\delta^* \colon S \times A^\star \to S$ of a deterministic automaton is a monoid action—see Exercise 1.4.1—for the free monoid structure on $A^\star$ from Exercise 1.4.3.

2.2.7. Note that the function spaces S^S carries a monoid structure given by composition. Show that the iterated transition function δ^* for a deterministic automaton, considered as a monoid homomorphism $A^\star \to S^S$, is actually obtained from δ by freeness of $A^\star$—as described in Exercise 1.4.3.

2.2.8. Consider a very simple differential equation of the form $\frac{df}{dy} = -Cf$, where $C \in \mathbb{R}$ is a fixed positive constant. The solution is usually described as $f(y) = f(0) \cdot e^{-Cy}$. Check that it can be described as a monoid action $\mathbb{R} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$, namely $(x, y) \mapsto xe^{-Cy}$.

2.2.9. Let $[0, 1] \subseteq \mathbb{R}$ be the unit interval of real numbers. For an arbitrary function $\varphi \colon X \to [0, 1]$ one writes $\mathsf{supp}(\varphi)$ for the "support" of φ, *i.e.* for the set $\{x \in X \mid \varphi(x) \neq 0\}$. A **probability distribution** is a function $\varphi \colon X \to [0, 1]$ satisfying $\sum_{x \in X} \varphi(x) = 1$. Such a distribution φ describes for each $x \in X$ the chance $\varphi(x)$ that it will occur. We shall write $\mathcal{D}(X)$ for the set of such probability distributions on X, and $\mathcal{D}_{\mathrm{fin}}(X) \hookrightarrow \mathcal{D}(X)$ for the subset of distributions with finite support.

(i) A distribution $\varphi \colon X \to [0, 1]$ may be understood as a formal sum $\sum_{x \in X} a_x x$ where $a_x = \varphi(x) \in [0, 1]$. For a function $f \colon X \to Y$ we put $\mathcal{D}(f)(\varphi) = \sum_{x \in X} \varphi(x) f(x)$. Check that $\mathcal{D}(f)(\varphi)$ is again a probability distribution.
(ii) Prove that both $\mathcal{D}$ and $\mathcal{D}_{\mathrm{fin}}$ are functors $\mathbf{Sets} \to \mathbf{Sets}$.
(iii) Note that $\mathcal{D}$-coalgebras may be called **probabilistic transition systems**. They are also known as Markov chains. Investigate what coalgebra homomorphisms are.
(iv) In analogy with (2.13), call a function $X \to \mathcal{D}(Y)$ a probabilistic relation $X \to Y$. Prove that one obtains a category in this way—analogously to the category **REL** from Example 1.4.2 (iv)—in which composition of $R \colon X \to \mathcal{D}(Y)$ and $S \colon Y \to \mathcal{D}(Z)$ is given by the formula:

$$(S \circ R)(x)(z) \quad = \quad \sum_{y \in Y} R(x, y) \cdot S(y, z). \tag{2.15}$$

The identity map $X \to \mathcal{D}(X)$, given by $x \mapsto 1x = (\lambda y.\, \mathsf{if}\ y = x\ \mathsf{then}\ 1\ \mathsf{else}\ 0)$, is sometimes called Dirac's delta.
(v) Use the previous point to show that Markov chains $S \to \mathcal{D}(S)$ form a monoid—in analogy with the situation for powersets in Exercise 1.1.2.

[For a systematic description of various forms of probabilistic transitions systems in terms of such coalgebras, see [235, 183, 34]. The distribution functor $\mathcal{D}$ is in many ways similar to the powerset functor $\mathcal{P}$. Later, much of their common structure will be described in terms of monads [86].]

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

(i) Prove that the product $V \times W$ of (the underlying sets of) two vector spaces V and W is at the same time a product and a coproduct in **Vect**—the same phenomenon as in the category of monoids, see Exercise 2.1.6. Show also that the singleton space 1 is both an initial and a final object. And notice that an element x in a vector space V may be identified with a linear map $\mathbb{R} \to V$.
(ii) A **linear dynamical system** [153] consists of three vector spaces: S for the state space, A for input, and B for output, together with three linear transformations: an input map $G \colon A \to S$, a dynamics $F \colon S \to S$, and an output map $H \colon S \to B$. Note how the first two maps can be combined via cotupleing into one transition function $S \times A \to S$, as used for deterministic automata. Because of the possibility of decomposing the transition function in this linear case into two maps $A \to S$ and $S \to S$, these systems are called *decomposable* by Arbib and Manes [15].

[But this transition function $S \times A \to S$ is not bilinear (*i.e.* linear in each argument separately), so it does not give rise to a map $S \to S^A$ to the vector space S^A of linear transformations from A to S. Hence we do not have a purely coalgebraic description $S \to S^A \times B$ in this linear setting.]

(iii) For a vector space A, consider, in the notation of [15], the subset of infinite sequences:

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

Equip the set $A^{\S}$ with a vector space structure, such that the insertion map $\mathsf{in}\colon A \to A^{\S}$, defined as $\mathsf{in}(a) = (a, 0, 0, 0, \ldots)$, and shift map $\mathsf{sh}\colon A^{\S} \to A^{\S}$, given as $\mathsf{sh}(\alpha) = (0, \alpha(0), \alpha(1), \ldots)$, are linear transformations.
[This vector space $A^{\S}$ may be understood as the space of polynomials over A in one variable. It can be defined as the infinite coproduct $\coprod_{n\in\mathbb{N}} A$ of $\mathbb{N}$-copies of A—which is also called a *copower*, and written as $\mathbb{N} \cdot A$, see [167, III, 3]. It is the analogue in **Vect** of the set of finite sequences B^* for $B \in \mathbf{Sets}$. This will be made precise in Exercise 2.4.6.]

(iv) Consider a linear dynamical system $A \xrightarrow{G} S \xrightarrow{F} S \xrightarrow{H} B$ as above, and show that the analogue of the behaviour $A^\star \to B$ for deterministic automata (see also [17, 6.3]) is the linear map $A^{\S} \to B$ defined as

$$(a_0, a_2, \ldots, a_n, 0, 0, \ldots) \quad \longmapsto \quad \sum_{i\leq n} HF^iGa_i$$

This is the standard behaviour formula for linear dynamical systems, see *e.g.* [153, 18]. [This behaviour map can be understood as starting from the "default" initial state $0 \in S$. If one wishes to start from an arbitrary initial state $x \in S$, one gets the formula

$$(a_0, a_2, \ldots, a_n, 0, 0, \ldots) \quad \longmapsto \quad HF^{(n+1)}x + \sum_{i\leq n} HF^iGa_i. \qquad]$$

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

2.3 Final coalgebras

In the previous chapter we have seen the special rôle that is played by the final coalgebra $A^\infty \to 1 + (A \times A^\infty)$ of sequences, both for defining functions into sequences and for reasoning about them—with "coinduction". Here we shall define finality in general for coalgebras, and investigate this notion more closely—which leads for example to language acceptance by automata, see Corolary 2.3.6 (ii). This general definition will allow us to use coinductive techniques for arbitrary final coalgebras. The underlying theme is that in final coalgebras there is no difference between states and their behaviours.

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

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

We shall start in full categorical generality—building on Definition 1.4.5—but we quickly turn to concrete examples in the category of sets. Examples of final coalgebras in other categories—like categories of domains or of metric spaces—may be found in many places, like [79, 80, 81, 225, 3, 233, 73, 72, 117], but also in Section 4.5.

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

there is a unique homomorphism $\mathsf{beh}_c\colon X \to Z$ of coalgebras, as in:

$$\begin{array}{ccc} F(X) & \overset{F(\mathsf{beh}_c)}{\dashrightarrow} & F(Z) \\ {\scriptstyle c}\uparrow & & \uparrow{\scriptstyle \zeta} \\ X & \underset{\mathsf{beh}_c}{\dashrightarrow} & Z \end{array}$$

The dashed notation is often used for uniqueness. What we call a behaviour map beh_c is sometimes called an *unfold* or a *coreduce* of c.

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

This unique existence is in fact all we need about final coalgebras. What these coalgebras precisely look like—what their elements are in $\mathbf{Sets}$—is usually not relevant. Nevertheless, in order to become more familiar with this topic of final coalgebras, we shall describe several examples concretely.

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

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

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

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

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

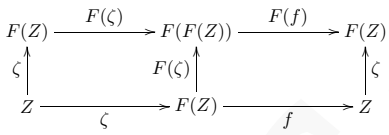
2.3.3. Lemma. *A final coalgebra $\zeta\colon Z \to F(Z)$ is necessarily an isomorphism $\zeta\colon Z \xrightarrow{\cong} F(Z)$.*

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

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

The aim is to show that this f is the inverse of ζ. We first consider the composite $f \circ \zeta\colon Z \to Z$, and show that it is the identity. We do so by first observing that the identity

$Z \to Z$ is the unique homomorphism $\zeta \to \zeta$. Therefore, it suffices to show that $f \circ \zeta$ is also a homomorphism $\zeta \to \zeta$. This follows from an easy diagram chase:



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

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

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

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

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

Proof. A standard result of Cantor (proved by so-called diagonalisation, see *e.g.* [63, Theorem 15.10] or [49, Theorem 1.10]) says that there cannot be an injection $\mathcal{P}(X) \rightarrowtail X$, for any set X. This excludes a final coalgebra $X \xrightarrow{\cong} \mathcal{P}(X)$. ❑

As we shall see later in this section, the *finite* powerset functor does have a final coalgebra. But first we shall look at some easier examples. The following result, occurring for example in [22, 206, 128], is simple but often useful.

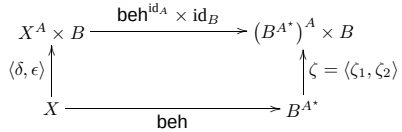
2.3.5. Proposition. *Fix two sets A and B, and consider the polynomial functor $\mathrm{id}^A \times B$ whose coalgebras are deterministic automata. The final coalgebra of this functor is given by the set of behaviour functions $B^{A^\star}$, with structure:*

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

given by:

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

Proof. We have to show that for an arbitrary coalgebra / deterministic automaton $\langle \delta, \epsilon \rangle\colon X \to X^A \times B$ there is a unique homomorphism of coalgebras $X \to B^{A^\star}$. For this we take of course the behaviour function $\mathsf{beh}\colon X \to B^{A^\star}$ from the previous section, defined in (2.14) by $\mathsf{beh}(x) = \lambda\sigma \in A^\star.\, \epsilon(\delta^*(x, \sigma))$. We have to prove that it is the unique function making the following diagram commute.



We prove commutation first. It amounts to two points, see also Lemma 2.2.2 (i).

$$\begin{array}{rcll} \big(\zeta_1 \circ \mathsf{beh}\big)(x)(a) & = & \zeta_1(\mathsf{beh}(x))(a) & \\ & = & \lambda\sigma.\, \mathsf{beh}(x)(a \cdot \sigma) & \\ & = & \lambda\sigma.\, \mathsf{beh}\big(\delta(x)(a)\big)(\sigma) & \text{see Exercise 2.2.5 (i)} \\ & = & \mathsf{beh}\big(\delta(x)(a)\big) & \\ & = & \mathsf{beh}^{\mathrm{id}_A}\big(\delta(x)\big)(a) & \\ & = & \big(\mathsf{beh}^{\mathrm{id}_A} \circ \delta\big)(x)(a) & \\ (\zeta_2 \circ \mathsf{beh})(x) & = & \mathsf{beh}(x)(\langle\rangle) & \\ & = & \epsilon(\delta^*(x, \langle\rangle)) & \\ & = & \epsilon(x). & \end{array}$$

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

$$\begin{array}{rcll} f(x)(\langle\rangle) & = & \zeta_2(f(x)) & \\ & = & \epsilon(x) & \text{since } f \text{ is a homomorphism} \\ & = & \mathsf{beh}(x)(\langle\rangle) & \\ f(x)(a \cdot \sigma) & = & \zeta_1(f(x))(a)(\sigma) & \\ & = & f\big(\delta(x)(a)\big)(\sigma) & \text{since } f \text{ is a homomorphism} \\ & = & \mathsf{beh}\big(\delta(x)(a)\big)(\sigma) & \text{by induction hypothesis} \\ & = & \mathsf{beh}(x)(a \cdot \sigma) & \text{see Exercise 2.2.5 (i).} \end{array}$$ ❑

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

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

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

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

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

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

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

given by:

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

where L_a is the so-called a-derivative, introduced by Brzozowski [48], and defined as:

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

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

Both these final coalgebras $A^{\mathbb{N}}$ and $\mathcal{P}(A^\star)$ are studied extensively by Rutten, see [220, 217, 218], see also Example 3.4.5 later on.

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

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

Here we write f' for the derivative of f. The induced coalgebra homomorphism $\mathsf{beh}_d\colon \mathcal{A} \to \mathbb{R}^{\mathbb{N}}$ maps an analytic function f to the sequence $(f(0), f'(0), f''(0), \ldots, f^{(n)}(0), \ldots)$, *i.e.* to its Taylor series expansion, used in:

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

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

The source of this "coinductive view on calculus" is [192]. It contains a further elaboration of these ideas. Other coalgebraic "next" or "tail" operations are also studied as derivatives in [217, 218].

Corolary 2.3.4 implies that there is no final coalgebra for the non-deterministic automata functor $\mathcal{P}(\mathrm{id})^A \times B$. However if we restrict ourselves to the finite powerset functor $\mathcal{P}_{\mathrm{fin}}(-)$ there is a final coalgebra. At this stage we shall be very brief, basically limiting ourselves to the relevant statement. It is a special case of Theorem 2.3.9, the proof of which will be given later, in Section 4.4.

2.3.8. Proposition (After [29]). *Let A and B be arbitary sets. Consider the finite Kripke polynomial functor $\mathcal{P}_{\mathrm{fin}}(id)^A \times B$ whose coalgebras are image finite non-deterministic automata. This functor has a final coalgebra.*

So far in this section we have seen several examples of final coalgebras. One might wonder which polynomial functors possess a coalgebra. The following result gives an answer. Its proof will be postponed until later in Section 4.4, because it requires some notions that have not been introduced yet.

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

As argued before, one does not need to know what a final coalgebra looks like in order to work with it. Its states coincide with its behaviours, so a purely behaviouristic view is justified: unique existence properties are sufficiently strong to use it as a black box. See for instance Exercise 2.3.2 below.

A good question raised explicitly in [216] is: which kind of functions can be defined with coinduction? Put another way: is there, in analogy with the collection of recursively defined functions (on the natural numbers), a reasonable collection of *corecursively* defined functions? This question is still largely open.

2.3.1 Beyond sets

So far we have concentrated on functors $F\colon \mathbf{Sets} \to \mathbf{Sets}$ on the category of sets and functions. Indeed, most of the examples in this book will arise from such functors. It is important however to keep the broader (categorical) perspective in mind, and realise that coalgebras are also of interest in other universes. Good examples appear in Section 4.5 where traces of suitable coalgebras are described via coalgebras in the category **REL** of sets with relations as morphisms. At this stage we shall consider a single example in the category $\mathbf{Sp}$ of topological spaces and continuous functions between them.

2.3.10. Example. The set $2^{\mathbb{N}} = \mathcal{P}(\mathbb{N})$ (with $2 = \{0, 1\}$) of infinite sequences of bits (or subsets of $\mathbb{N}$) carries a topology yielding the well-known "Cantor space", see [224, Subsection 2.3]. Alternatively, this set can be represented as the intersection of a descending chain of intervals (2^n separate pieces at stage n) of the real interval $[0, 1]$, see any textbook on topology (for instance [46]). The basic opens of $2^{\mathbb{N}}$ are the subsets $\uparrow \sigma = \{\sigma \cdot \tau \mid \tau \in 2^{\mathbb{N}}\}$ of infinite sequences starting with σ, for $\sigma \in 2^\star$ a finite sequence.

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

A brief argument goes as follows. Corollary 2.3.6 (i) tells that the set of streams $2^{\mathbb{N}}$ is the final coalgebra of the endofunctor $X \mapsto X \times 2$ on $\mathbf{Sets}$. There is an obvious isomorphism $X \times 2 \cong X + X$ of sets (see Exercise 2.1.7), which is actually also an isomorphism of topological spaces if one considers the set 2 with the discrete topology (in which every subset is open), see Exercise 2.3.5 for more details.

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

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

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

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

Again, this map is continuous.

Exercises

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

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

2.3.3. Recall the decimal representation coalgebra $\mathsf{nextdec}\colon [0, 1) \to 1 + (\{0, 1, \ldots, 9\} \times [0, 1))$ from Example 1.2.2, with its behaviour map $\mathsf{beh}_{\mathsf{nextdec}}\colon [0, 1) \to \{0, 1, \ldots, 9\}^\infty$. Prove that this behaviour map is a split mono: there is a map e in the reverse direction with $e \circ \mathsf{beh}_{\mathsf{nextdec}} = \mathrm{id}_{[0,1)}$.

[The behaviour map is not an isomorphism, because both 5 and $49999\cdots$, considered as sequences in $\{0,1,\ldots,9\}^\infty$, represent $\frac{1}{2} \in [0,1)$. See other representations as continued fractions in [193] or [187] which do yield isomorphisms.]

2.3.4. This exercise is based on [128, Lemma 5.4].

(i) Fix three sets A, B, C, and consider the simple polynomial functor

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

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

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

Once such functions $\varphi \in Z$ hit C, they keep this value in C. Here we write A^+ for the subset of $A^\star$ of non-empty finite sequences. The associated coalgebra structure $\zeta\colon Z \xrightarrow{\cong} (C + (Z \times B))^A$ given by:

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

(ii) Check that the fact that the set B^∞ of sequences is the final coalgebra of the functor $X \mapsto 1 + (X \times B)$ is a special case of this.

(iii) Generalise the result in (i) to functors of the form

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

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

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

where $A = A_1 + \cdots + A_n$, $B = B_1 + \cdots + B_n$, and $C = C_1 + \cdots + C_n$.

(iv) Show how classes like in (1.7) fit into this last result.
[*Hint.* Use that $S + (S \times E) \cong (S \times 1) + (S \times E) \cong S \times (1 + E)$, using distributivity from Excercise 2.1.7.]

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

(i) Check that the head $A^{\mathbb{N}} \to A$ and tail $A^{\mathbb{N}} \to A^{\mathbb{N}}$ operations are continuous.

(ii) Prove that the functor $A \times (-)\colon \mathbf{Sp} \to \mathbf{Sp}$ has a final coalgebra.

(iii) Show that in the special case where A carries the discrete topology (in which every subset is open) the product topology on $A^{\mathbb{N}}$ is given by basic opens $\uparrow \sigma = \{\sigma \cdot \tau \mid \tau \in A^{\mathbb{N}}\}$, for $\sigma \in A^\star$ like in Example 2.3.10.

2.3.6. (i) Note that the assignment $A \mapsto A^{\mathbb{N}}$ yields a functor $\mathbf{Sets} \to \mathbf{Sets}$.

(ii) Prove the general result: consider a category $\mathbb{C}$ with a functor $F\colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ in two variables. Assume that for each object $A \in \mathbb{C}$, the functor $F(A,-)\colon \mathbb{C} \to \mathbb{C}$ has a final coalgebra $Z_A \xrightarrow{\cong} F(A, Z_A)$. Prove that the mapping $A \mapsto Z_A$ extends to a functor $\mathbb{C} \to \mathbb{C}$.

2.4 Algebras

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

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

This section will introduce the categorical definition of algebras for a functor, in analogy with coalgebras of a functor in Definition 1.4.5. It will briefly discuss initiality for algebras, as dual of finality, and will illustrate that it amounts to ordinary induction. Also, it will mention several possible ways of combining algebras and coalgebras. A systematic approach to such combinations using distributive laws will appear in Chapter 6.

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

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

(i) An F-**algebra**, or just **algebra** for F, consists of a "carrier" object $X \in \mathbb{C}$ together with a morphism $a\colon F(X) \to X$, often called the constructor, or operation.

(ii) A **homomorphism of algebras**, or a **map of algebras**, or an **algebra map**, from one algebra $a\colon F(X) \to X$ to another coalgebra $b\colon F(Y) \to Y$ consists of a morphism $f\colon X \to Y$ in $\mathbb{C}$ which preserves the operations:

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

This yields a category, for which we shall write $\mathbf{Alg}(F)$.

(iii) An **initial** F-algebra is an initial object in $\mathbf{Alg}(F)$: it is an algebra $\alpha\colon F(A) \to A$ such that for any F-algebra $b\colon F(X) \to X$ there is a unique homomorphism of algebras $\mathsf{int}_b\colon A \to X$, which we see as an interpretation map.

What we call an interpretation map int_b is also called a *fold* or a *reduce* of b.

Certain maps can be seen both as algebra and as coalgebra. For example, a map $S \times A \to S$ is an algebra—of the functor $X \mapsto X \times A$—but can also be regarded as a coalgebra $S \to S^A$, after Currying (2.8). But for other maps there is more clarity: maps $A \to S$ are algebras, which can be used to construct elements in S from elements in A. And maps $S \to A$ are coalgebras, which provide observations in A about elements in S.

The functor F in the above definition corresponds to what is traditionally called a signature. In simple form a signature consists of a number of operations op_i, say for $1 \leq i \leq n$, each with their own arity $k_i \in \mathbb{N}$. An algebra for such a signature is then a set S with interpretations $[\![\, \mathsf{op}_i \,]\!]\colon S^{k_i} \to S$. Using the coproduct correspondence (2.5), they may be combined to a single cotuple operation,

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

forming an algebra of the simple polynomial functor $X \mapsto X^{k_1} + \cdots + X^{k_n}$. This functor thus captures the number and types of the operations—that is, the signature.

A rule of thumb is: data types are algebras, and state-based systems are coalgebras. But this does not always give a clear-cut distinction. For instance, is a stack a data type or does it have a state? In many cases however, this rule of thumb works: natural numbers are

algebras (as we are about to see), and machines are coalgebras. Indeed, the latter have a state that can be observed and modified.

Initial algebras are special, just like final coalgebras. Initial algebras (in $\mathbf{Sets}$) can be built as so-called term models: they contain everything that can be built from the operations themselves, and nothing more. Similarly, we saw that final coalgebras consist of observations only. The importance of initial algebras in computer science was first emphasised in [93]. For example, if F is the signature functor for some programming language, the initial algebra $F(P) \to P$ may be considered as the set of programs, and arbitrary algebras $F(D) \to D$ may be seen as denotational models. Indeed, the resulting unique homomorphism $[\![-]\!]\colon P \to D$ is the semantical interpretation function. It is *compositional* by construction, because it commutes with the operations of the programming language.

In this context of program language semantics, a final coalgebra is also understood as a canonical operational model of behaviours. The unique homomorphism to the final coalgebra may be seen as an operational interpretation function which is *fully abstract*, in the sense that objects have the same interpretation if and only if they are observationally indistinguishable. This relies on Theorem 3.4.1, see [233, 231, 232] for more information. Compositionality for such models in final coalgebras is an issue, see Section **??**.

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

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

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

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

2.4.3. Example (Natural numbers). According to Peano, the most basic operations on the natural numbers $\mathbb{N}$ are zero $\mathsf{0} \in \mathbb{N}$ and successor $\mathsf{S}\colon \mathbb{N} \to \mathbb{N}$. Using that an element of $\mathbb{N}$ can be described as an arrow $1 \to \mathbb{N}$, together with the coproduct correspondence (2.5), these two maps can be combined into an algebra

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

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

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

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

The initiality property of $[\mathsf{0},\mathsf{S}]\colon 1 + \mathbb{N} \xrightarrow{\cong} \mathbb{N}$ is strong enough to prove all of Peano's axioms. This was first shown in [78], see also [168], or [94], where the initiality is formulated as "natural numbers object". As an example, we shall consider the familiar induction rule: for a predicate $P \subseteq \mathbb{N}$,

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

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

$$\begin{array}{ccc} 1 & \overset{\mathsf{0}}{\dashrightarrow} & P \\ & {\scriptstyle \mathsf{0}}\searrow & \downarrow{\scriptstyle i} \\ & & \mathbb{N} \end{array} \qquad\qquad \begin{array}{ccc} P & \overset{\mathsf{S}}{\dashrightarrow} & P \\ {\scriptstyle i}\downarrow & & \downarrow{\scriptstyle i} \\ \mathbb{N} & \xrightarrow[\mathsf{S}]{} & \mathbb{N} \end{array}$$

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

$$\begin{array}{ccccc} 1 + \mathbb{N} & \xrightarrow{\mathsf{id}_1 + j} & 1 + P & \xrightarrow{\mathsf{id}_1 + i} & 1 + \mathbb{N} \\ {\scriptstyle [\mathsf{0},\mathsf{S}]}\downarrow & & \downarrow{\scriptstyle [\mathsf{0},\mathsf{S}]} & & \downarrow{\scriptstyle [\mathsf{0},\mathsf{S}]} \\ \mathbb{N} & \xrightarrow{j} & P & \xrightarrow{i} & \mathbb{N} \\ & \searrow^{\mathrm{id}_{\mathbb{N}}} & & & \end{array}$$

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

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

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

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

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

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

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

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

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

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

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

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

It turns out that many such functional programs can be defined elegantly via initiality (and also finality). Moreover, via uniqueness one can establish various properties about these programs. This has developed into an area of its own which is usually referred to as the Bird-Meertens formalism—with its own peculiar notation and terminology, see [42] for an overview.

As we have seen in Example 2.4.3, an inductive definition of a function $f\colon \mathbb{N} \to X$ on the natural numbers requires two functions $a\colon 1 \to X$ and $g\colon X \to X$. A **recursive** definition allows for an additional parameter, via a function $h\colon X \times \mathbb{N} \to X$, determining $f(n+1) = h(f(n), n)$. The next result shows that recursion can be obtained via induction. A more general approach via comonads may be found in [234]. An alternative way to add parameters to induction occurs in Exercise 2.5.3.

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

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

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

$$\begin{array}{rcl} \pi_2 \circ k \circ \alpha & = & \pi_2 \circ h' \circ F(k) \\ & = & \alpha \circ F(\pi_2) \circ F(k) \\ & = & \alpha \circ F(\pi_2 \circ k). \end{array}$$

Hence we take $f = \pi_1 \circ k$. ❑

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

- Joyal and Moerdijk [152] introduce the notion of a *Zermelo-Fraenkel algebra*, or *ZF-algebra*, with two operations for union and singleton. The usual system of Zermelo-Fraenkel set theory can then be characterised as the free ZF-algebra. Such a characterisation can be extended to ordinals.

- Fiore, Plotkin and Turi [71] describe how variable binding—such as $\lambda x.\, M$ in the lambda calculus—can also be captured via initial algebras, namely of suitable functors on categories of presheaves. An alternative approach, also via initiality, is described by Gabbay and Pitts [83], using set theory with atoms (or *urelements*).

In the remainder of this section we shall survey several ways to combine algebras and coalgebras. Such combinations are needed in descriptions of more complicated systems involving data as well as behaviour. Later on, in Chapter 6, we shall study bialgebras more systematically.

Additionally, so-called "binary methods" are problematic in a coalgebraic setting. These are (algebraic) operations like $X \times X \to X$ in which the state space X occurs multiple times (positively) in the domain. The name "binary method" comes from object-oriented programming, where the status of such methods is controversial, due to the typing problems they can cause, see [47]. They are also problematic in coalgebra, because they cannot be described as a coalgebra $X \to F(X)$. However, one may ask whether it makes sense in system theory to have an operation acting on two states, so that the problematic character of their representation need not worry us very much. But see Exercise 2.4.7 for a possible way to handle binary methods in a coalgebraic manner.

2.4.1 Bialgebras

A **bialgebra** consists of an algebra-coalgebra pair $F(X) \to X \to G(X)$ on a common state space X, where $F, G\colon \mathbb{C} \to \mathbb{C}$ are two endofunctors on the same category $\mathbb{C}$. We shall investigate them more closely in Chapter 6 using so-called distributive laws, following the approach of [232, 231, 33].

Such structures are used in [40] to distinguish certain observer operations as coalgebraic operations within algebraic specification, and to use these in a duality between reachability and observability (following the result of Kalman, see Exercise 2.5.12).

2.4.2 Dialgebras

A **dialgebra** consist of a mapping $F(X) \to G(X)$ where F and G are two functors $\mathbb{C} \to \mathbb{C}$. This notion generalises both algebras (for $G = \mathrm{id}$) and coalgebras (for $F = \mathrm{id}$). It was introduced in [106], and further studied for example in [74] in combination with "laws" as a general concept of equation. In [202] a collection of such dialgebras $F_i(X) \to G_i(X)$ is studied, in order to investigate the communalities between algebra and coalgebra, especially related to invariants and bisimulations.

2.4.3 Hidden algebras

Hidden algebra is introduced in [92] as the "theory of everything" in software engineering, combining the paragdigms of object-oriented, logic, constraint and functional programming. A hidden algebra does not formally combine algebras and coalgebras, like in bi-/di-algebras, but it uses an algebraic syntax to handle essentially coalgebraic concepts, like behaviour and observational equivalence. A key feature of the syntax is the partition of sorts (or types) into visible and hidden ones. Typically, data structures are visible, and states are hidden. Elements of the visible sorts are directly observable and comparable,

but observations about elements of the hidden sorts can only be made via the visible ones. This yields a notion of behavioural equivalence—first introduced by Reichel—expressed in terms of equality of contexts of visible sort. This is in fact bisimilarity, from a coalgebraic perspective, see [172, 52].

Because of the algebraic syntax of hidden algebras one cannot represent typically coalgebraic operations. But one can mimic them via subsorts. For instance, a coalgebraic operation $X \longrightarrow 1 + X$ can be represented as a "partial" function $S \longrightarrow X$ for a subsort $S \subseteq X$. Similarly, an operation $X \longrightarrow X + X$ can be represented via two operations $S_1 \longrightarrow X$ and $S_2 \longrightarrow X$ for subsorts $S_1, S_2 \subseteq X$ with $S_1 \cup S_2 = X$ and $S_1 \cap S_2 = \emptyset$. This quickly gets out of hand for more complicated operations, like the methods meth_i from (1.7), so that the naturality of coalgebraic representations is entirely lost. On the positive side, hidden algebras can handle binary methods without problems—see also Exercise 2.4.7.

2.4.4 Coalgebras as algebras

In general, there is reasonable familiarity with algebra, but not (yet) with coalgebra. Therefore, people like to understand coalgebras as if they were algebras. Indeed, there are many connections. For example, in [29] it is shown that quite often the final coalgebra $Z \xrightarrow{\cong} F(Z)$ of a functor F can be understood as a suitable form of completion of the initial algebra $F(A) \xrightarrow{\cong} A$ of the same functor. Examples are the extended natural numers $\mathbb{N} \cup \{\infty\}$ from Exercise 2.4.1 below, as completion of the initial algebra $\mathbb{N}$ of the functor $X \mapsto 1 + X$. And the set A^∞ of finite and infinite sequences as completion of the initial algebra $A^\star$ with finite sequences only, of the functor $X \mapsto 1 + (A \times X)$.

Also, since a final coalgebra $\zeta\colon Z \xrightarrow{\cong} F(Z)$ is an isomorphism, one can consider its inverse $\zeta^{-1}\colon F(Z) \xrightarrow{\cong} Z$ as an algebra. This happens for example in the formalisation of "coinductive types" (final coalgebras) in the theorem prover Coq [32] (and used for instance in [59, 121]). However, this may lead to rather complicated formulations of coinduction, distracting from the coalgebraic essentials. Therefore we like to treat coalgebra as a field of its own, and not in an artificial way as part of algebra.

Exercises

2.4.1. Check that the set $\mathbb{N} \cup \{\infty\}$ of extended natural numbers is the final coalgebra of the functor $X \mapsto 1 + X$, as a special case of finality of A^∞ for $X \mapsto 1 + (A \times X)$. Use this fact to define appropriate addition and multiplication operations $(\mathbb{N} \cup \{\infty\}) \times (\mathbb{N} \cup \{\infty\}) \to \mathbb{N} \cup \{\infty\}$, see [216].

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

2.4.3. Show, dually to Exercise 2.1.13, that finite products $(1, \times)$ in a category $\mathbb{C}$ give rise to finite products in a category $\mathbf{Alg}(F)$ of algebras of a functor $F\colon \mathbb{C} \to \mathbb{C}$.

2.4.4. Define addition $+\colon \mathbb{N}\times\mathbb{N} \to \mathbb{N}$, multiplication $\cdot\colon \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ and exponentation $(-)^{(-)}\colon \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ by initiality.
[*Hint.* Use the correspondence (2.8) and define these operations as function $\mathbb{N} \to \mathbb{N}^{\mathbb{N}}$. Alternatively, one can use Exercise 2.5.3 from the next section.]

2.4.5. Complete the proof of Proposition 2.4.5.

2.4.6. This exercise illustrates the analogy between the set $A^\star$ of finite sequences in $\mathbf{Sets}$ and the vector space $A^\S$ in **Vect**, following Exercise 2.2.10.

(i) For an arbitrary set A, consider the functor $1 + (A \times \mathrm{id})\colon \mathbf{Sets} \to \mathbf{Sets}$. Show that the set $A^\star$ of finite sequences of elements of A is the initial algebra of this functor.

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

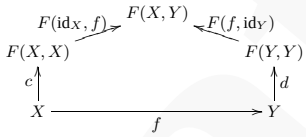
$$\begin{array}{rcll} 1 + (A \times X) & \cong & A \times X & \text{because } 1 \in \mathbf{Vect} \text{ is initial object} \\ & \cong & A + X & \text{because } \times \text{ and } + \text{ are the same in } \mathbf{Vect} \end{array}$$

Prove that the initial algebra of the resulting functor $A + \mathrm{id}\colon \mathbf{Vect} \to \mathbf{Vect}$ is the vector space $A^\S$ with insertion and shift maps $\mathsf{in}\colon A \to A^\S$ and $\mathsf{sh}\colon A^\S \to A^\S$ defined in Exercise 2.2.10 (iii).

(iii) Check that the behaviour formula from Exercise 2.2.10 (iv) for a system $A \xrightarrow{F} X \xrightarrow{G} X \xrightarrow{H} B$ is obtained as $H \circ \mathsf{int}_{[F,G]}\colon A^\S \to B$ using initiality.

(iv) Show that the assignment $A \mapsto A^\S$ yields a functor $\mathbf{Vect} \to \mathbf{Vect}$.
[*Hint.* Actually, this is a special case of the dual of Exercise 2.3.6.]

2.4.7. Suppose we have a "binary method", say of the form $m\colon X \times X \times A \longrightarrow 1 + (X \times B)$. There is a well-known trick from [79] to use a functorial description also in such cases, namely by separating positive and negative occurrences. This leads to a functor of the form $F\colon \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbb{C}$, which in this case would be $(Y, X) \mapsto (1 + (X \times B))^{Y \times A}$.
In general, for a functor $F\colon \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbb{C}$ define an F-coalgebra to a map of the form $c\colon X \to F(X, X)$. A homormorphism from $c\colon X \to F(X, X)$ to $d\colon Y \to F(Y, Y)$ is then a map $f\colon X \to Y$ in $\mathbb{C}$ making the following pentagon commute.



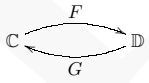
(i) Elaborate what this means for the above example $(Y, X) \mapsto (1 + (X \times B))^{Y \times A}$.

(ii) Prove that these coalgebras and their homomorphisms form a category.

[Such generalised coalgebras are studied systematically in [229].]

2.5 Adjunctions, cofree coalgebras, behaviour-realisation

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



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

We start with "baby-adjunctions", namely Galois connections. Consider two posets C and D as categories, with monotone functions $f\colon C \to D$ and $g\colon D \to C$ between them, in opposite direction. They can be regarded as functors by Example 1.4.4 (ii). These functions form a **Galois connection** if for each $x \in C$ and $y \in D$ one has $f(x) \leq y$ in D if and only if $x \leq g(y)$ in C. We like to write this as a bijective correspondence:

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

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

With these correspondences it is easy to show that the left adjoint f preserves joins $\vee$ that exist in C, *i.e.* that $f(x_1 \vee x_2) = f(x_1) \vee f(x_2)$. This is done by showing that $f(x_1 \vee x_2)$ is the least upperbound in D of $f(x_1)$ and $f(x_2)$: for any $y \in D$,

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

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

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

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

$$\dfrac{F(X) \longrightarrow Y \text{ in } \mathbb{D}}{X \longrightarrow G(Y) \text{ in } \mathbb{C}}$$

which are "natural" in X and Y. In this case one says that F is the left adjoint, and G the right adjoint.

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

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

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

There are several equivalent ways to formulate the concept of an adjunction, using for example unit and counit natural transformations, or freeness (for left adjoints) or cofreeness (for right adjoints). We shall describe one alternative, namely the unit-and-counit formulation, in Exercise 2.5.5, and refer the interested reader to [167, Chapter IV] for more information. At this stage we are mainly interested in a workable formulation.

We continue with an elementary example. It illustrates that adjunctions involve canonical translations back and forth, which can be understood as minimal (or, more technically: free) constructions, for left adjoints, and as maximal (or cofree) for right adjoints. Earlier, Exercise 1.4.3 described the set of finite sequences $A^\star$ as *free* monoid on a set A. Below in Exercise 2.5.1 this will be rephrased in terms of an adjunction; it gives a similar minimality phenomenon.

2.5.2. Example (From sets to preorders). Recall from Example 1.4.2 (iii) the definition of the category **PreOrd** with preorders $(X, \leq)$ as objects—where $\leq$ is a reflexive and transitive order on X—and monotone functions between them as morphisms. There is then an obvious forgetful functor $U\colon \mathbf{PreOrd} \to \mathbf{Sets}$, given by $(X, \leq) \mapsto X$. It maps a preorder to its underlying set and forgets about the order. This example shows that the forgetful functor U has both a left and a right adjoint.

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

$$\dfrac{D(A) = (A, \mathrm{Eq}(A)) \xrightarrow{\;f\;} (X, \leq) \quad \text{in } \mathbf{PreOrd}}{A \xrightarrow[\;f\;]{} X = U(X, \leq) \quad \text{in } \mathbf{Sets}}$$

This yields an adjunction $D \dashv U$.

The right adjoint $I\colon \mathbf{Sets} \to \mathbf{PreOrd}$ equips a set A with the "indiscrete" preorder $I(A) = (A, \top_{A \times A})$ with "truth" relation $\top_A = \{(a, a') \mid a, a' \in A\}$ in which all elements of A are related. Then, for a preorder $(X, \leq)$, any function $X \to A$ is automatically a monotone function $(X, \leq) \to (A, \top_{A \times A})$. Hence we again have trivial correspondences:

$$\dfrac{U(X, \leq) = X \xrightarrow{\;f\;} A \quad \text{in } \mathbf{Sets}}{(X, \leq) \xrightarrow[\;f\;]{} (A, \top_{A \times S}) = I(A) \quad \text{in } \mathbf{PreOrd}}$$

yielding an adjunction $U \dashv I$.

This situation "discrete $\dashv$ forgetful $\dashv$ indiscrete" is not uncommon, see Excercise 2.5.4 below.

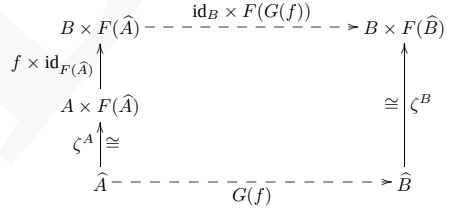
Next we shall consider examples of adjunctions in the context of coalgebras. The first result describes "cofree" coalgebras: it gives a canonical construction of a coalgebra from an arbitrary set. Its proof relies on Theorem 2.3.9, which is as yet unproven. The proof shows that actually checking that one has an adjunction can be quite a bit of work. Indeed, the notion of adjunction combines much information.

2.5.3. Proposition. *For a finite Kripke polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$, the forgetful functor $U\colon \mathbf{CoAlg}(F) \to \mathbf{Sets}$ has a right adjoint.*

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

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

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



Clearly, $\zeta_2^B \circ G(f) = F(G(f)) \circ \zeta_2^A$, so that $G(f)$ is a homomorphism of coalgebras $G(A) \to G(B)$, as required. By uniqueness one shows that G preserves identities and compositions. This requires a bit of work but is not really difficult.

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

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

That is:

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

It is obtained as follows.

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

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

  Then $\zeta_2^A \circ \overline{g} = F(\overline{g}) \circ c$, so that $\overline{g}$ is a homomorphism of coalgebras $c \to G(A)$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In the reverse direction $\mathcal{P}_{\mathrm{fin}}(X) \to X^\star$ one can always choose a way to turn a finite set into a list, but there is no natural way to do this. Natural transformations describe uniform mappings, such given for instance by terms, see Exercise 2.5.14.

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

We shall consider another example of an adjunction which is typical in a coalgebraic setting, namely a so-called *behaviour-realisation* adjunction. Such adjunctions were first recognised in the categorical analysis of mathematical system theory, see [88, 89, 90]. Here we give a simple version using the deterministic automata introduced in Section 2.2. This requires two extensions of what we have already seen: (1) homomorphisms between these automata which allow variation in the input and output sets, and (2) automata with initial states.

2.5.5. Definition. We write **DA** for the category with:

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

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

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

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

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

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

The first two equations express that h is a coalgebra homomorphism from the automaton $\langle \mathrm{id}_X^f \circ \delta, g \circ \epsilon \rangle \colon X \to X^C \times D$, translated via (f, g), to the automaton $\langle \delta', \epsilon' \rangle \colon X \to X^C \times D$. Here we use the exponent notation id_X^f for functions from (2.9). The third equation simply says that the initial state is preserved.

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

2.5.6. Definition. Form the category **DB** with

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

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

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

with for $\sigma \in C^\star$,

$$g(\varphi(f^\star(\sigma))) = \psi(\sigma)$$

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

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

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

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

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

$$\mathbf{DA} \quad \mathcal{B} \dashv \mathcal{R} \quad \mathbf{DB}$$

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

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

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

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

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

$$\begin{array}{ccc} X^C \times D & \xrightarrow{h^{\mathrm{id}_C} \times \mathrm{id}_D} & (D^{C^\star})^C \times D \\ {\scriptstyle \mathrm{id}_X^f \times g} \uparrow & & \uparrow {\scriptstyle \cong\, \zeta} \\ X^A \times B & & \\ {\scriptstyle \langle \delta, \epsilon \rangle} \uparrow & & \\ X & \xrightarrow[h]{} & D^{C^\star} \end{array}$$

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

Several alternative versions of this behaviour-realisation adjunction for deterministic automata are described in the exercises below. Such adjunctions have also been studied in more abstract settings, see for example [24] and [155].

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

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

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

Both these definitions yield a so-called symmetric monoidal structure on the category **DA** of deterministic automata. Similar structure can be defined on the associated category

DB of behaviours, in such a way that the behaviour functor $\mathcal{B}\colon \mathbf{DA} \to \mathbf{DB}$ preserves this structure. Thus, complex behaviour can be obtained from more elementary building blocks.

There is a line of research studying such "process categories" with operations that are commonly used in process calculi as appropriate categorical structure, see for instance [115, 156, 111, 222, 161, 240, 27].

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

Exercises

2.5.1. Recall from Exercise 1.4.3 that the assignment $A \mapsto A^\star$ gives a functor $(-)^\star\colon \mathbf{Sets} \to \mathbf{Mon}$ from sets to monoids. Show that this functor is left adjoint to the forgetful functor $\mathbf{Mon} \to \mathbf{Sets}$ which maps a monoid $(M, +, 0)$ to its underlying set M and forgets the monoid structure.

2.5.2. This exercise describes "strength" for endofunctors on $\mathbf{Sets}$. In general, this is a useful notion in the theory of datatypes [54, 55] and of computations [182].
Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be an arbitrary functor. Consider for sets X, Y the strength map

$$F(X) \times Y \xrightarrow{\mathsf{st}_{X,Y}} F(X \times Y) \quad \text{given by} \quad (u, y) \longmapsto F(\lambda x \in X.\, (x, y))(u)$$

(i) Prove that this yields a natural transformation $F(-) \times (-) \overset{\mathsf{st}}{\Longrightarrow} F((-) \times (-))$, where both the domain and codomain are functors $\mathbf{Sets} \times \mathbf{Sets} \to \mathbf{Sets}$.
(ii) Sometimes strength is formulated via a map

$$F(Y^X) \xrightarrow{\mathsf{st}'} F(Y)^X$$

as:

$$\begin{aligned} \mathsf{st}' &= \Lambda(F(\mathsf{ev}) \circ \mathsf{st}) \\ &= \lambda v \in F(Y^X).\, \lambda x \in X.\, F(\lambda f \in Y^X.\, f(x))(v). \end{aligned}$$

Show that st' is natural too, and that st can also be obtained from st'. Hence the two formulations are equivalent—in general, in Cartesian closed categories.

2.5.3. The following strengthening of induction is sometimes called "induction with parameters". It is different from recursion, which also involves an additional parameter, see Proposition 2.4.5.
Assume a functor F with a strength natural transformation as in the previous exercise, and with initial algebra $\alpha\colon F(A) \xrightarrow{\cong} A$. Let P be a set (or object) for parameters. Prove that for each map $h\colon F(X) \times P \to X$ there is a unique $f\colon A \times P \to X$ making the following diagram commute.

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

[*Hint.* First turn h into a suitable algebra $h'\colon F(X^P) \to X^P$.]

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

2.5.5. Assume two functors $F\colon \mathbb{C} \to \mathbb{D}$ and $G\colon \mathbb{D} \to \mathbb{C}$ in opposite directions, with natural transformations $\eta\colon \mathrm{id}_{\mathbb{C}} \Rightarrow GF$ and $\varepsilon\colon FG \Rightarrow \mathrm{id}_{\mathbb{D}}$. Define functions $\psi\colon \mathbb{D}(F(X), Y) \longrightarrow \mathbb{C}(X, G(Y))$ by $\psi(f) = G(f) \circ \eta_X$.

(i) Check that such ψ's are natural.
(ii) Prove that they are isomorphisms if and only if the following **triangular identities** hold.

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

2.5.6. A morphism $m\colon X' \to X$ in a category $\mathbb{D}$ is called a **monomorphism**, (or **mono**, for short) written as $m\colon X' \rightarrowtail X$, if for each parallel pair of arrows $f, g\colon Y \to X'$, $m \circ f = m \circ g$ implies $f = g$.
(i) Prove that the monomorphisms in $\mathbf{Sets}$ are precisely the injective functions.
(ii) Let $G\colon \mathbb{D} \to \mathbb{C}$ be a right adjoint. Show that if m is a monomorphism in $\mathbb{D}$, then so is $G(m)$ in $\mathbb{C}$.
Dually, an **epimorphism** (or **epi**, for short) in $\mathbb{C}$ is an arrow written as $e\colon X' \twoheadrightarrow X$ such that for all maps $f, g\colon X \to Y$, if $e \circ f = e \circ g$, then $f = g$.
(iii) Show that the epimorphims in $\mathbf{Sets}$ are the surjective functions.
[*Hint.* For an epi $X \twoheadrightarrow Y$, choose two appropriate maps $Y \to 1 + Y$.]
(iv) Prove that left adjoints preserve epimorphisms.

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

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

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

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

2.5.9. Recall the distribution functors $\mathcal{D}, \mathcal{D}_{\mathrm{fin}}\colon \mathbf{Sets} \to \mathbf{Sets}$ from Exercise 2.2.9. Check that there are natural transformations:
(i) $\eta\colon \mathrm{id} \Rightarrow \mathcal{D}$ and $\eta\colon \mathrm{id} \Rightarrow \mathcal{D}_{\mathrm{fin}}$ both given by Dirac's delta $x \mapsto 1x = (\lambda y.\, \mathsf{if}\ y = x\ \mathsf{then}\ 1\ \mathsf{else}\ 0)$.
(ii) $\mathsf{supp}\colon \mathcal{D} \Rightarrow \mathcal{P}$ and $\mathsf{supp}\colon \mathcal{D}_{\mathrm{fin}} \Rightarrow \mathcal{P}_{\mathrm{fin}}$ where the support $\mathsf{supp}(\varphi)$ of a distribution $\varphi\colon X \to [0, 1]$ is the set $\{x \mid \varphi(x) \neq 0\}$.

2.5.10. Consider two adjoint endofunctors as in:

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

Prove that we then get an isomorphism of categories:

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

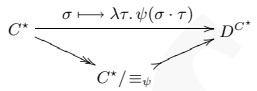
between the associated categories of algebras and coalgebras.
[Remark: as noted in [16, Theorem 5.7], when $\mathbb{C} = \mathbf{Sets}$ the only such adjunctions $F \dashv G$ are product-exponent adjunctions $X \times (-) \dashv (-)^X$ as in (2.8). The argument goes as follows. In $\mathbf{Sets}$, each object A can be written as coproduct $\coprod_{a \in A} 1$ of singletons. A left adjoint F must preserve such coproducts, so that $F(A) \cong \coprod_{a \in A} F(1) \cong F(1) \times A$. But then $G(-) \cong F(1) \Rightarrow (-)$, by uniqueness of adjoints.]

2.5.11. A deterministic automaton $\langle \delta, \epsilon \rangle\colon X \to X^A \times B$ is called **observable** if its behaviour function $\mathsf{beh} = \lambda x.\, \lambda \sigma.\, \epsilon(\delta^*(x, \sigma))\colon X \to B^{A^\star}$ from Proposition 2.3.5 is injective. Later, in Corollary 3.4.3 we shall see that this means that bisimilar states are equal.
If this automaton comes equipped with an initial state $x_0 \in X$ one calls the automaton **reachable** if the function $\delta^*(x_0, -)\colon A^\star \to X$ from Section 2.2 is surjective. This means that every state can be reached from the initial state x_0 via a suitable sequence of inputs. The automaton is called **minimal** if it is both observable and reachable.
The realisation construction $\mathcal{R}\colon \mathbf{DB} \to \mathbf{DA}$ from Propostion 2.5.7 clearly yields an observable automaton since the resulting behaviour function is the identity. An alternative

construction, the so-called **Nerode realisation**, gives a minimal automaton. It is obtained from a behaviour $\psi\colon C^\star \to D$ as follows. Consider the equivalence relation $\equiv_\psi \subseteq C^\star \times C^\star$ defined by:

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

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



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

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

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

(ii) Check that the behaviour functor $\mathcal{B}\colon$ **DA** $\to$ **DB** from Proposition 2.5.7 restricts to a functor $\mathcal{B}\colon$ **RDA** $\to$ **RDB**, and show that it has Nerode realisation $\mathcal{N}$ yields a functor **RDA** $\to$ **RDB** in the opposite direction.

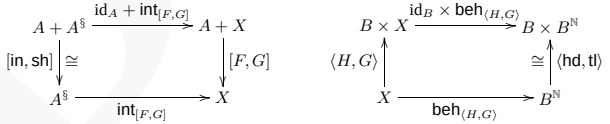
(iii) Prove that there is an adjunction $\mathcal{B} \dashv \mathcal{N}$.

(iv) Let **MDA** be the "subcategory" of **RDA** with minimal automata as objects. Check that the adjunction in the previous point restricts to an equivalence of categories **MDA** $\simeq$ **RDB**. Thus, (states of) minimal automata are in fact (elements of) final coalgebras. [This result comes from [89, 90], see also [8].]

2.5.12. This exercise and the next one continue the description of linear dynamical systems from Exercise 2.2.10. Here we look at the duality between reachability and observability. First a preliminary result.

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

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



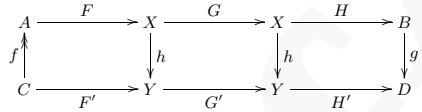
Similarly, call this system **observable** if the induced map $\mathsf{beh}_{\langle H,G\rangle}\colon X \to B^{\mathbb{N}}$ on the right is injective (equivalently, a monomorphism in **Vect**). And call the system **minimal** if it is both reachable and observable.

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

[Kalman's duality result [153, Chapter 2] is now an easy consequence in *finite dimensional* vector spaces, where the adjoint operator $(-)^*$ makes **Vect** isomorphic to **Vect**$^{\mathrm{op}}$—where V^* is of course the "dual" vector space of linear maps to the underlying field. This result says that $A \xrightarrow{F} X \xrightarrow{G} X \xrightarrow{H} B$ is reachable if and only if $B^* \xrightarrow{H^*} X^* \xrightarrow{G^*} X^* \xrightarrow{F^*} A^*$ is observable. See also [16]. There is also a duality result for bialgebras in [40].]

2.5.13. This exercise sketches an adjunction capturing Kalman's minimal realisation [153, Chapter 10] for linear dynamical systems, in analogy with the Nerode realisation in Exercise 2.5.11. This categorical description is based on [15, 16].

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

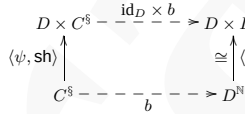


Note that f is required to be a surjection/epimorphism.

Also, there is a category **RLB** with linear maps $\varphi\colon A^\S \to B$ as objects. A morphism $\left(A^\S \xrightarrow{\varphi} B\right) \longrightarrow \left(C^\S \xrightarrow{\psi} D\right)$ is a pair of linear maps $f\colon C \twoheadrightarrow A$ and $g\colon B \to D$ with $g \circ \varphi \circ f^\S = \psi$—where $f^\S$ results from the functoriality of $(-)^\S$, see Exercise 2.4.6 (iv).

(i) Show that the behaviour formula from Exercises 2.2.10 (iv) and 2.4.6 (iii) yields a behaviour functor $\mathcal{B}\colon$ **RLDS** $\to$ **RLB**, given by $(F, G, H) \longmapsto H \circ \mathsf{int}_{[F,G]}$.

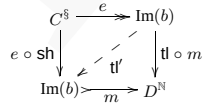
(ii) Construct a functor $\mathcal{K}\colon$ **RLB** $\to$ **RLDS** in the reverse direction in the following way. Assume a behaviour $\psi\colon C^\S \to D$, and form the behaviour map $b = \mathsf{beh}_{\langle\psi,\mathsf{sh}\rangle}\colon C^\S \to D^{\mathbb{N}}$ below, using the finality from the previous exercise:



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

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

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



Hence one can define a linear dynamical system as:

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

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

2.5.14. This exercise describes the so-called "terms-as-natural-transformations" view which originally stems from [170]. It is elaborated in a coalgebraic context in [166].

Let $H\colon \mathbf{Sets} \to \mathbf{Sets}$ be a (not necessarily polynomial) functor, with free algebras, given by a left adjoint F to the forgetful functor $U\colon \mathbf{Alg}(H) \to \mathbf{Sets}$. Let X be an arbitrary set, whose elements are considered as variables. Elements of the carrier $UF(X)$ of the free algebra on X can then be seen as terms containing free variables from X. Show that there is a bijective correspondence:

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

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

Chapter 3

Bisimulations

The operation of a coalgebra gives us information about its states. It may allow us to observe certain things, and it may allow us to "modify states", or to "move to successor states". Typically for coalgebras, we can observe and modify, but we have no means for constructing new states. The behaviour of a state x is all that we can observe about x, either directly or indirectly (via its successor states). This behaviour is written as $\mathsf{beh}(x)$, where beh is the unique map to the final coalgebra (if any), as introduced in Definition 2.3.1.

In this situation it may happen that two states have the same behaviour. In that case we cannot distinguish them with the operations (of the coalgebras) that we have at our disposal. The two states need not be equal then, since the operations may only give limited access to the state space, and certain aspects that may not be observable. When two states x, y are observationally indistinguishable, they are called *bisimilar*. This is written as $x \leftrightarroweq y$.

The bisimilarity relation, for a given coalgebra (or pair of coalgebras), is introduced as the union of all bisimulations. A bisimulation is a relation on state spaces, which is maintained by coalgebra transitions and leads to equal observations. Bisimulations were first introduced in [189] for automata, as mutual simulations—building on an earlier notion of simulation between programs [178]. Park proved that if the initial states of two deterministic automata are related by a bisimulation, then they accept the same sets of inputs (see also Corollary 3.4.4 below). Indeed, bisimulations form a crucial tool for stepwise reasoning—like in induction arguments.

Bisimilarity is the main topic of the present chapter. It is introduced—like congruence—via a technique called relation lifting. In a related manner the notion of invariance will arise in the next chapter via predicate lifting. The first part of this chapter concentrates on some basic properties of relation lifting and of bisimulation relations; these properties will be used frequently. The coinduction proof principle in Section 3.4 is a basic result in the theory of coalgebras. It says that two states have the same behaviour if and only if they are contained in a bisimulation relation. Coinduction via bisimulations is illustrated in a simple process calculus in Section 3.5.

3.1 Relation lifting, bisimulations and congruences

This section will introduce the technique of relation lifting from [116, 117] and use it to define the notions of bisimulation for coalgebras, and congruence for algebras. Many elementary results about relation lifting are provided. Alternative ways for introducing bisimulations will be discussed later in Section 3.3.

We start by motivating the need for relation lifting. Consider a sequence coalgebra $c\colon X \to 1 + (A \times X)$, like in Section 1.2. A bisimulation for this coalgebra is a relation $R \subseteq X \times X$ on its state space which is "closed under c". What this means is that if R holds for states x, y, then either both x and y have no successor states (*i.e.* $c(x) = \kappa_1(*) = c(y)$),

or they both have successor states which are again related by R and their observations are the same: $c(x) = \kappa_2(a, x')$, $c(y) = \kappa_2(b, y')$, with $a = b$ and $R(x', y')$.

One way to express such closure properties uniformly is via a "lifting" of R from a relation on X to a relation R' on $1 + (A \times X)$ so that this closure can be expressed simply as:

$$R(x, y) \implies R'(c(x), c(y)).$$

This idea works if we take $R' \subseteq \big(1 + (A \times X)\big) \times \big(1 + (A \times X)\big)$ to be

$$R' = \{\kappa_1(*), \kappa_1(*)\} \cup \{(\kappa_2(a, x), \kappa_2(b, y)) \mid a = b \wedge R(x, y)\}.$$

The general idea of relation lifting applies to a polynomial functor F. It is a transformation of a relation $R \subseteq X \times X$ to a relation $R' \subseteq F(X) \times F(X)$, which will be defined by induction on the structure of F. We shall use the notation $\mathrm{Rel}(F)(R)$ for R' above. Briefly, the lifted relation $\mathrm{Rel}(F)(R)$ uses equality on occurrences of constants A in F, and R on occurrences of the state space X, as suggested in:

$$\begin{array}{rcl} F(X) & = & \boxed{\cdots \quad X \quad \cdots \quad A \quad \cdots \quad X \quad \cdots} \\ \mathrm{Rel}(F)(R) & = & \quad\; R \qquad\quad \| \qquad\quad R \\ F(X) & = & \boxed{\cdots \quad X \quad \cdots \quad A \quad \cdots \quad X \quad \cdots} \end{array}$$

Actually, it will be convenient to define relation lifting slightly more generally, and to allow different state spaces. Thus, it applies to relations $R \subseteq X \times Y$, and yields a relation $\mathrm{Rel}(F)(R) \subseteq F(X) \times F(Y)$.

3.1.1. Definition (Relation lifting). Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor, and let X, Y be arbitrary sets. The mapping $\mathrm{Rel}(F)$ which sends a relation $R \subseteq X \times Y$ to a "lifted" relation $\mathrm{Rel}(F)(R) \subseteq F(X) \times F(Y)$ is defined by induction on the structure of the functor F, following the points in Definition 2.2.1.

(1) If F is the identity functor, then

$$\mathrm{Rel}(F)(R) = R.$$

(2) If F is a constant functor $Z \mapsto A$, then

$$\mathrm{Rel}(F)(R) = \mathrm{Eq}(A) = \{(a, a) \mid a \in A\}.$$

(3) If F is a product $F_1 \times F_2$, then

$$\begin{array}{rcl} \mathrm{Rel}(F)(R) & = & \{((u_1, u_2), (v_1, v_2)) \mid \mathrm{Rel}(F_1)(R)(u_1, v_1) \wedge \\ & & \qquad \mathrm{Rel}(F_2)(R)(u_2, v_2)\}. \end{array}$$

(4) If F is a coproduct $F_1 + F_2$, then

$$\begin{array}{rcl} \mathrm{Rel}(F)(R) & = & \{(\kappa_1(u), \kappa_1(v)) \mid \mathrm{Rel}(F_1)(R)(u, v)\} \cup \\ & & \{(\kappa_2(u), \kappa_2(v)) \mid \mathrm{Rel}(F_2)(R)(u, v)\}. \end{array}$$

(5) If F is an exponent G^A, then

$$\mathrm{Rel}(F)(R) = \{(f, g) \mid \forall a \in A.\, \mathrm{Rel}(G)(R)(f(a), g(a))\}.$$

(6) If F is a powerset $\mathcal{P}(G)$, then

$$\begin{array}{rcl} \mathrm{Rel}(F)(R) & = & \{(U, V) \mid \forall u \in U.\, \exists v \in V.\, \mathrm{Rel}(G)(R)(u, v) \wedge \\ & & \qquad \forall v \in V.\, \exists u \in U.\, \mathrm{Rel}(G)(R)(u, v)\}. \end{array}$$

This same formula will be used in case F is a *finite* powerset $\mathcal{P}_{\mathrm{fin}}(G)$.

(7) If F is a list $G^\star$, then

$$\mathrm{Rel}(F)(R) = \{(\langle u_1, \ldots, u_n\rangle, \langle v_1, \ldots, v_n\rangle) \mid \forall i \leq n.\, \mathrm{Rel}(G)(R)(u_i, v_i)\}.$$

In the beginning of Section 3.3 we shall see that relation lifting can also be defined directly via images. The above inductive definition may seem more cumbersome, but gives us a better handle on the different cases. Also, it better emphasises the relational aspects of lifting, and the underlying logical infrastructure (such as finite conjunctions and disjunctions, and universal and existential quantification). This is especially relevant in more general settings, such as in [117].

Relation lifting w.r.t. a functor is closely related to so-called logical relations. These are collections of relations $(R_\sigma)_\sigma$ indexed by types σ, in such a way that $R_{\sigma\to\tau}$ and $R_{\sigma\times\tau}$ are determined by R_σ and R_τ. Similarly, we use collections of relations $(\mathrm{Rel}(F)(R))_F$ indexed by polynomial functors F, which are also structurally determined. Logical relations were originally introduced in the context of semantics of (simply) typed lambda calculus ([227, 82, 198]), see [181, Chapter 8] for an overview. They are used for instance for definability, observational equivalence and data refinement.

In the next section we shall see various elementary properties of relation lifting. But first we show what it is used for: bisimulation for coalgebras, and congruence for algebras. The definitions we use are *generic* or *polytypic*, in the sense that they apply uniformly to (co)algebras of an arbitrary polynomial functor.

3.1.2. Definition. Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor.

(i) A **bisimulation** for coalgebras $c\colon X \to F(X)$ and $d\colon Y \to F(Y)$ is a relation $R \subseteq X \times Y$ which is "closed under c and d":

$$(x, y) \in R \implies (c(x), d(y)) \in \mathrm{Rel}(F)(R).$$

for all $x \in X$ and $y \in Y$. Equivalently:

$$R \subseteq (c \times d)^{-1}(\mathrm{Rel}(F)(R)),$$

or, by (2.12),

$$\coprod_{c\times d}(R) \subseteq \mathrm{Rel}(F)(R).$$

(ii) A **congruence** for algebras $a\colon F(X) \to X$ and $b\colon F(Y) \to Y$ is a relation $R \subseteq X \times Y$ which is also "closed under a and b":

$$(u, v) \in \mathrm{Rel}(F)(R) \implies (a(u), b(v)) \in R.$$

That is:

$$\mathrm{Rel}(F)(R) \subseteq (a \times b)^{-1}(R) \qquad \text{or} \qquad \coprod_{a\times b}(\mathrm{Rel}(F)(R)) \subseteq R.$$

Often we are interested in bisimulations $R \subseteq X \times X$ on a *single* coalgebra $c\colon X \to F(X)$. We then use the definition with $d = c$. Similarly for congruences.

Notice that we only require that a congruence is closed under the (algebraic) operations, and not that it is an equivalence relation. This minor deviation from standard terminology is justified by the duality we obtain between bisimulations and congruences. We shall use the following explicit terminology.

3.1.3. Definition. A **bisimulation equivalence** is a bisimulation on a single coalgebra which is an equivalence relation. Similarly, a **congruence equivalence** is a congruence on a single algebra which is an equivalence relation.

We continue with several examples of the notions of bisimulation and congruence for specific functors.

Bisimulations for deterministic automata

Consider a deterministic automaton $\langle \delta, \epsilon \rangle \colon X \to X^A \times B$. As we have seen in Section 2.2, it is a coalgebra for the functor $F = \mathrm{id}^A \times B$. Relation lifting for this functor yields for a relation $R \subseteq X \times X$ a new relation $\mathrm{Rel}(F)(R) \subseteq \big(X^A \times B\big) \times \big(X^A \times B\big)$, given by:

$$\mathrm{Rel}(F)(R)\Big((f_1, b_1), (f_2, b_2)\Big) \iff \forall a \in A.\, R(f_1(a), f_2(a)) \wedge b_1 = b_2.$$

Thus, a relation $R \subseteq X \times X$ is a bisimulation w.r.t. the (single) coalgebra $\langle \delta, \epsilon \rangle \colon X \to X^A \times B$ if, for all $x, y \in X$,

$$R(x, y) \implies \mathrm{Rel}(F)(R)\Big((\delta(x), \epsilon(x)), (\delta(y), \epsilon(y))\Big).$$

I.e.

$$R(x, y) \implies \forall a \in A.\, R(\delta(x)(a), \delta(y)(a)) \wedge \epsilon(x) = \epsilon(y).$$

That is, in transition notation:

$$R(x, y) \implies \begin{cases} x \xrightarrow{a} x' \wedge y \xrightarrow{a} y' \text{ implies } R(x', y') \\ x \downarrow b \wedge y \downarrow c \text{ implies } b = c. \end{cases}$$

Thus, once two states are in a bisimulation R, they remain in R and give rise to the same direct observations. This makes them observationally indistinguishable.

Bisimulations for non-deterministic automata

Next, consider a non-deterministic automaton $\langle \delta, \epsilon \rangle \colon X \to \mathcal{P}(X)^A \times B$, as coalgebra for the functor $F = \mathcal{P}(\mathrm{id})^A \times B$. Relation lifting for this functor is slightly more complicated: it sends a relation $R \subseteq X \times X$ to the relation $\mathrm{Rel}(F)(R) \subseteq \big(\mathcal{P}(X)^A \times B\big) \times \big(\mathcal{P}(X)^A \times B\big)$ given by:

$$\begin{aligned} \mathrm{Rel}(F)(R)\big((f_1, b_1), (f_2, b_2)\big) \iff \forall a \in A.\ & \forall u \in f_1(a).\, \exists v \in f_2(a).\, R(u, v) \wedge \\ & \forall v \in f_2(a).\, \exists u \in f_1(a).\, R(u, v) \\ & \wedge b_1 = b_2. \end{aligned}$$

Thus, $R \subseteq X \times X$ is a bisimulation if for all $x, y \in X$ with $R(x, y)$,

- $x \xrightarrow{a} x'$ implies there is a y' with $y \xrightarrow{a} y'$ and $R(x', y')$;
- $y \xrightarrow{a} y'$ implies there is an x' with $x \xrightarrow{a} x'$ and $R(x', y')$;
- $x \downarrow b$ and $y \downarrow c$ implies $b = c$.

This corresponds to the standard notion of bisimulation used in the theory of automata / transition systems.

Congruences for monoids

Recall that a monoid is a set M carrying an associative operation $+ \colon M \times M \to M$ with a unit element $0 \in M$. These two operations $+$ and 0, but not the relevant monoid equations, can be captured as an algebra

$$1 + (M \times M) \xrightarrow{\quad [0, +] \quad} M$$

of the functor $F(X) = 1 + (X \times X)$. Relation lifting for F is described by

$$\mathrm{Rel}(F)(R) \;=\; \{(\kappa_1(*), \kappa_1(*))\} \cup \{(\kappa_2(x, x'), \kappa_2(y, y')) \mid R(x, y) \wedge R(x', y')\}$$

Hence a relation $R \subseteq M \times M$ on the carrier of the monoid is a congruence if:

$$\mathrm{Rel}(F)(R)(u, v) \implies R([0, +](u), [0, +](v))$$

This amounts to:

$$R(0, 0) \qquad \text{and} \qquad R(x, y) \wedge R(x', y') \implies R(x + x', y + y')$$

Thus a congruence, like a bisimulation, is closed under the operations.

Congruences in a binary induction proof principle

We have already discussed the usual "unary" induction proof principle for natural numbers in Example 2.4.3, expressed in terms of predicates, which are assumed to be closed under the operations. Later, in Section 4.1 we shall encounter it in full generality, stating that every invariant on an initial algebra is the truth predicate.

There is also a less well-known binary version of the induction proof principle, expressed in terms of congruences. It was first formulated as such for the natural numbers in [219], and further generalised in [117]. It also appeared in the derivations of induction and coinduction principles in [197] in the context of a formal logic for parametric polymorphism.

At this stage we only formulate this binary version, and postpone the proof. It can be given in various ways, see Exercises 3.3.3 and 4.2.2, but requires some properties of relation lifting which are still to come.

3.1.4. Theorem (Binary induction proof principle). *Every congruence on an initial algebra contains the equality relation.*

This binary version of induction is the dual of a coinduction principle, see Corollary 3.4.2.

Bisimulations as congruences

The so-called structural operational semantics (SOS) introduced by Plotkin is a standard technique in the semantics of programming languages to define the operational behaviour of programs. The latter are seen as elements of the initial algebra $F(\mathsf{Prog}) \xrightarrow{\cong} \mathsf{Prog}$ of a suitable functor F describing the signature of operations of the programming language. A transition relation is then defined on top of the set of programs Prog, as the least relation closed under certain rules. This transition structure may be understood as coalgebra $\mathsf{Prog} \to G(\mathsf{Prog})$, for an appropriate functor G—which is often the functor $\mathcal{P}(\mathrm{id})^A$ for labeled transition systems, see [232, 231]; the transition structure is then given by transitions $p \xrightarrow{a} q$ describing an a-step between programs $p, q \in \mathsf{Prog}$.

The transition structure gives rise to certain equivalences for programs, like bisimilarity (see below), trace equivalence or other equivalences, see [87]. These equivalences are

typically bisimulation equivalences. An important issue in this setting is: are these bisimulation equivalences also *congruences* for the given algebra structure $F(\mathsf{Prog}) \xrightarrow{\cong} \mathsf{Prog}$. This is a basic requirement to make the equivalence a reasonable one for the kind of programs under consideration, because congruence properties are essential in reasoning with the equivalence. In this setting given by a bialgebra $F(\mathsf{Prog}) \to \mathsf{Prog} \to G(\mathsf{Prog})$, the two fundamental notions of this section (bisimulation and congruence) are thus intimately related. This situation will be investigated further in Chapter 6 in relation to distributive laws.

It is a whole area of research to establish suitable syntactic formats for SOS-rules guaranteeing that certain bisimulation equivalences are congruences. See [99] for a basic reference. We shall use a more categorical perspective, first in Section 3.5, and later in Chapter 6, following [232, 231, 33].

3.1.5. Definition. Let $X \xrightarrow{c} F(X)$ and $Y \xrightarrow{d} F(Y)$ be two coalgebras of a polynomial functor F. The **bisimilarity** relation $\leftrightarroweq$ is the union of all bisimulations:

$$x \leftrightarroweq y \iff \exists R \subseteq X \times Y.\, R \text{ is a bisimulation for } c \text{ and } d, \text{ and } R(x,y)$$

As a result of Proposition 3.2.5 (iii) in the next section, this union is a bisimulation itself, so that $\leftrightarroweq$ can be characterised as the greatest bisimulation.

Sometimes we write ${}_c\!\leftrightarroweq_d$ for $\leftrightarroweq$ to make the dependence on the coalgebras c and d explicit.

Bisimilarity formalises the idea of observational indistinguishability. It will be an important topic in the remainder of this chapter.

Exercises

3.1.1. Unfold the definition of bisimulation for various kind of tree coalgebras, like $X \to 1 + (A \times X \times X)$ and $X \to (A \times X)^\star$.

3.1.2. Do the same for classes in object-oriented languages, see (1.7), described as coalgebras of a functor in Exercise 2.3.4 (iii).

3.1.3. Note that the operations of a vector space V (over $\mathbb{R}$), namely zero, addition, inverse, and scalar multiplication, can be captured as an algebra $1 + (V \times V) + V + (\mathbb{R} \times V) \longrightarrow V$. Investigate then what the associated notion of congruence is.

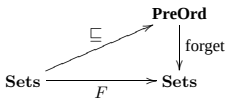
3.1.4. We have described relation lifting on a coproduct functor $F = F_1 + F_2$ in Definition 3.1.1 as:

$$\mathrm{Rel}(F_1 + F_2)(R) \;=\; \coprod_{\kappa_1 \times \kappa_1} (\mathrm{Rel}(F_1)(R)) \cup \coprod_{\kappa_2 \times \kappa_2} (\mathrm{Rel}(F_2)(R)).$$

Prove that it can also be defined in terms of products $\prod$ and intersection $\cap$ as:

$$\mathrm{Rel}(F_1 + F_2)(R) \;=\; \prod_{\kappa_1 \times \kappa_1} (\mathrm{Rel}(F_1)(R)) \cap \prod_{\kappa_2 \times \kappa_2} (\mathrm{Rel}(F_2)(R)).$$

3.1.5. In this text we concentrate on *bi*-simulations. There is also the notion of simulation, that can be defined via an order on a functor, see [230, 125]. For a functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$ such an order consists of a functor $\sqsubseteq$ as in the diagram below.



Given such an order we define "lax" relation lifting $\mathrm{Rel}_\sqsubseteq(F)$ as $R \mapsto \sqsubseteq \circ \mathrm{Rel}(F)(R) \circ \sqsubseteq$. A relation $R \subseteq X \times Y$ is then a **simulation** on coalgebras $X \xrightarrow{c} FX$, $Y \xrightarrow{d} FY$ if $R \subseteq (c \times d)^{-1}(\mathrm{Rel}_\sqsubseteq(F)(R))$. Similarity is then the union of all simulations.

(i) Investigate what it means to have an order as described in the above diagram.

(ii) Describe on the functor $L = 1 + (A \times (-))$ a "flat" order, and on the powerset functor $\mathcal{P}$ the inclusion order, as in the diagram. Check what the associated notions of simulation are.

(iii) Prove that similarity on the final coalgebra A^∞ of the functor L with order as in (ii) is the prefix order given by $\sigma \leq \tau$ iff $\sigma \cdot \rho = \tau$ for some $\rho \in A^\infty$, see [125, Example 5.7].

3.2 Properties of bisimulations

This section is slightly technical, and possibly also slightly boring. It starts by listing various elementary properties of relation lifting, and subsequently uses these properties to prove standard results about bisimulations and bisimilarity.

First there are three lemmas about relation lifting.

3.2.1. Lemma. *Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor. Relation lifting $\mathit{Rel}(F)$ w.r.t. F satisfies the following basic properties.*

(i) *It preserves the equality relation:*

$$\mathit{Rel}(F)(\mathit{Eq}(X)) \;=\; \mathit{Eq}(F(X))$$

(ii) *It commutes with reversal of relations:*

$$\mathit{Rel}(F)(R^{-1}) \;=\; \mathit{Rel}(F)(R)^{-1}$$

(iii) *It is monotone:*

$$R \subseteq S \implies \mathit{Rel}(F)(R) \subseteq \mathit{Rel}(F)(S)$$

(iv) *It preserves relation composition*

$$\mathit{Rel}(F)(R \circ S) \;=\; \mathit{Rel}(F)(R) \circ \mathit{Rel}(F)(S)$$

(v) *It preserves reflexivity, symmetry and transitivity; and thus, if R is an equivalence relation, then so is $\mathit{Rel}(F)(R)$.*

Proof. The first four statement (i)–(iv) are proved by induction on the structure of F, following the cases in Definition 3.1.1. The case in (iv) where F is an exponent G^A requires the Axiom of Choice (AC), as will be illustrated: assume, as induction hypothesis (IH), that the functor G preserves composition of relations, then so does the exponent G^A, since:

$$\begin{array}{ll}
& \mathrm{Rel}(G^A)(R \circ S)(f,g) \\
\iff & \forall a \in A.\, \mathrm{Rel}(G)(R \circ S)(f(a), g(a)) \\
\overset{\text{(IH)}}{\iff} & \forall a \in A.\, \big(\mathrm{Rel}(G)(R) \circ \mathrm{Rel}(G)(S)\big)(f(a), g(a)) \\
\iff & \forall a \in A.\, \exists z.\, \mathrm{Rel}(G)(R)(f(a), z) \wedge \mathrm{Rel}(G)(S)(z, g(a)) \\
\overset{\text{(AC)}}{\iff} & \exists h.\, \forall a \in \mathrm{Rel}(G)(R)(f(a), h(a)) \wedge \mathrm{Rel}(G)(S)(h(a), g(a)). \\
\iff & \exists h.\, \mathrm{Rel}(G^A)(R)(f,h) \wedge \mathrm{Rel}(G^A)(S)(h,g) \\
\iff & \big(\mathrm{Rel}(G^A)(R) \circ \mathrm{Rel}(G^A)(S)\big)(f,g).
\end{array}$$

The last statement (v) follows from the previous ones:

- If R is reflexive, *i.e.* $\mathrm{Eq}(X) \subseteq R$, then $\mathrm{Eq}(F(X)) = \mathrm{Rel}(F)(\mathrm{Eq}(X)) \subseteq \mathrm{Rel}(F)(R)$, so that $\mathrm{Rel}(F)(R)$ is also reflexive.
- If R is symmetric, *i.e.* $R \subseteq R^{-1}$, then $\mathrm{Rel}(F)(R) \subseteq \mathrm{Rel}(F)(R^{-1}) = \mathrm{Rel}(F)(R)^{-1}$, so that $\mathrm{Rel}(F)(R)$ is symmetric as well.

- If R is transitive, *i.e.* $R \circ R \subseteq R$, then $\mathrm{Rel}(F)(R) \circ \mathrm{Rel}(F)(R) = \mathrm{Rel}(F)(R \circ R) \subseteq \mathrm{Rel}(F)(R)$, so that $\mathrm{Rel}(F)(R)$ is also transitive. ❑

We proceed with a similar lemma, about relation lifting and (inverse and direct) images.

3.2.2. Lemma. *Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor again, and let $f\colon X \to Z$ and $g\colon Y \to W$ be arbitrary functions.*

(i) *Relation lifting commutes with inverse images: for $R \subseteq Z \times W$,*

$$\mathit{Rel}(F)((f \times g)^{-1}(R)) \quad = \quad (F(f) \times F(g))^{-1}(\mathit{Rel}(F)(R)).$$

(ii) *Relation lifting also commutes with direct images: for $R \subseteq X \times Y$,*

$$\mathit{Rel}(F)(\textstyle\coprod_{f\times g}(R)) \quad = \quad \coprod_{F(f)\times F(g)}(\mathit{Rel}(F)(R)).$$

Proof. Both equations are proved by induction on the structure of F. We leave (i) to the reader. Once (i) is established, it can be used to prove the direction $(\supseteq)$ of (ii), using the Galois connection relating direct and inverse images in (2.12):

$$\begin{array}{rcl} \coprod_{F(f)\times F(g)}(\mathrm{Rel}(F)(R)) & \subseteq & \mathrm{Rel}(F)(\coprod_{f\times g}(R)) \\ \iff \mathrm{Rel}(F)(R) & \subseteq & (F(f) \times F(g))^{-1}\mathrm{Rel}(F)(\coprod_{f\times g}(R)) \\ & = & \mathrm{Rel}(F)((f \times g)^{-1} \coprod_{f\times g}(R)). \end{array}$$

But this latter inclusion holds by monotonicity of relation lifting from Lemma 3.2.1 (iii), using that $R \subseteq (f \times g)^{-1} \coprod_{f\times g}(R)$.

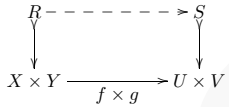
The inclusion $(\subseteq)$ of (ii) is proved by induction on the structure of the functor F. This requires the Axiom of Choice to handle the exponent functor case, like in the proof of point (iv) in the previous lemma. The powerset case $F = \mathcal{P}(G)$ is most complicated, and will be described in detail.

$$\begin{array}{ll} & \mathrm{Rel}(\mathcal{P}(G))(\coprod_{f\times g}(R))(U,V) \\ \iff & \forall x \in U.\, \exists y \in V.\, \mathrm{Rel}(G)(\coprod_{f\times g}(R))(x,y) \\ & \quad \wedge\, \forall y \in V.\, \exists x \in U.\, \mathrm{Rel}(G)(\coprod_{f\times g}(R))(x,y) \\ \overset{\text{(IH)}}{\iff} & \forall x \in U.\, \exists y \in V.\, \coprod_{G(f)\times G(g)} \mathrm{Rel}(G)(R)(x,y) \\ & \quad \wedge\, \forall y \in V.\, \exists x \in U.\, \coprod_{G(f)\times G(g)} \mathrm{Rel}(G)(R)(x,y) \\ \iff & \forall x \in U.\, \exists y \in V.\, \exists u,v.\, G(f)(u) = x \wedge G(g)(v) = y \wedge \mathrm{Rel}(G)(R)(u,v) \\ & \quad \wedge\, \forall y \in V.\, \exists x \in U.\, \exists u,v.\, G(f)(u) = x \wedge G(g)(v) = y \wedge \mathrm{Rel}(G)(R)(u,v) \\ \implies & \forall u \in U'.\, \exists v \in V'.\, \mathrm{Rel}(G)(R)(u,v) \\ & \quad \wedge\, \forall v \in V'.\, \exists u \in U'.\, \mathrm{Rel}(G)(R)(u,v), \quad \text{where} \\ & \qquad U' \;=\; \{u \mid G(f)(u) \in U \wedge \exists v.\, G(g)(v) \in V \wedge \mathrm{Rel}(G)(R)(u,v)\} \\ & \qquad V' \;=\; \{v \mid G(g)(v) \in V \wedge \exists u.\, G(f)(u) \in U \wedge \mathrm{Rel}(G)(R)(u,v)\} \\ \iff & \exists U', V'.\, \mathcal{P}(G(f))(U') = U \wedge \mathcal{P}(G(f))(V') = V \wedge \mathrm{Rel}(\mathcal{P}(G))(R)(U',V') \\ \iff & \coprod_{\mathcal{P}(G(f))\times\mathcal{P}(G(g))}(\mathrm{Rel}(\mathcal{P}(G))(R))(U,V). \end{array}$$

❑

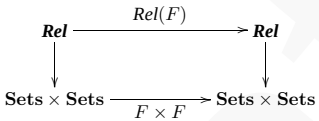
Below we show in a diagram why $\mathrm{Rel}(F)$ is called relation lifting. The term "relator" is often used in this context for the lifting of F, see for instance [230] (and Exercise 3.3.9, where a more general description of the situation is given). Additionally, the next result shows that bisimulations are coalgebras themselves. It involves a category **Rel** with relations as objects. This **Rel** should not be confused with the category **REL**, from Example 1.4.2 (iv), which has relations as morphisms.

3.2.3. Lemma. *Let us write **Rel** for the category with binary relations $R \subseteq X \times Y$ as objects. A morphism $(R \subseteq X \times Y) \longrightarrow (S \subseteq U \times V)$ consists of two functions $f\colon X \to U$ and $g\colon Y \to V$ with $R(x,y) \;\Rightarrow\; S(f(x), g(y))$ for all x, y. The latter amounts to the existence of the necessarily unique dashed map in:*



In this situation:

(i) *Relation lifting forms a functor:*



where the vertical arrows are the obvious forgetful functors.

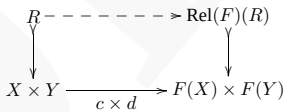
(ii) *A bisimulation is a $\mathrm{Rel}(F)$-coalgebra. Similarly, a congruence is a $\mathrm{Rel}(F)$-algebra.*

Proof. (i) We show that if $(f,g)\colon R \to S$ is a morphism in **Rel**—where $f\colon X \to U$ and $g\colon Y \to V$—then $(F(f), F(g))\colon \mathrm{Rel}(F)(R) \to \mathrm{Rel}(F)(S)$. Note that $R \subseteq (f \times g)^{-1}(S)$, and so by monotony and preservation of inverse images by relation lifting:

$$\mathrm{Rel}(F)(R) \quad \subseteq \quad \mathrm{Rel}(F)((f \times g)^{-1}(S)) \quad = \quad (F(f) \times F(g))^{-1}\mathrm{Rel}(F)(S).$$

This means that $(F(f), F(g))$ is a morphism $\mathrm{Rel}(F)(R) \to \mathrm{Rel}(F)(S)$ in **Rel**.

(ii) A $\mathrm{Rel}(F)$-coalgebra $R \to \mathrm{Rel}(F)(R)$ for $R \subseteq X \times Y$ consists of two underlying maps $c\colon X \to F(X)$ and $d\colon Y \to F(Y)$ with:



This says that R is a relation which is closed under the F-coalgebras c, d, *i.e.* that R is a bisimulation for c, d.

In the same way congruences are $\mathrm{Rel}(F)$-algebras. ❑

The next result lists several useful preservation properties of relation lifting.

3.2.4. Lemma. *Relation lifting $\mathit{Rel}(F)$ w.r.t. a polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$ preserves:*

(i) ***kernel** relations, given for an arbitrary function $f\colon X \to Y$ by:*

$$\begin{array}{rcl} \mathit{Ker}(f) & = & \{(x_1,x_2) \in X \times X \mid f(x_1) = f(x_2)\} \\ & = & (f \times f)^{-1}(\mathit{Eq}(Y)) \end{array}$$

in:

$$\mathit{Rel}(F)(\mathit{Ker}(f)) \quad = \quad \mathit{Ker}(F(f)).$$

(ii) ***graph*** *relations given for* $f\colon X \to Y$ *by:*

$$\begin{aligned} \mathit{Graph}(f) &= \{(x,y) \in X \times Y \mid f(x) = y\} \\ &= (f \times \mathit{id}_Y)^{-1}(\mathit{Eq}(Y)) \end{aligned}$$

in:

$$\mathit{Rel}(F)(\mathit{Graph}(f)) = \mathit{Graph}(F(f)).$$

(iii) ***images*** *of tuples: for* $X \xleftarrow{f} Z \xrightarrow{g} Y$,

$$\begin{aligned} \mathit{Im}(\langle f,g\rangle) &= \{(x,y) \in X \times Y \mid \exists z \in Z.\, f(z) = x \wedge g(z) = y\} \\ &= \coprod_{f\times g}(\mathit{Eq}(Z)) \end{aligned}$$

in:

$$\mathit{Rel}(F)(\mathit{Im}(\langle f,g\rangle)) = \mathit{Im}(\langle F(f), F(g)\rangle).$$

(iv) ***pullback*** *relations of spans: for* $X \xrightarrow{f} Z \xleftarrow{g} Y$,

$$\begin{aligned} \mathit{Pb}(f,g) &= \{(x,y) \in X \times Y \mid f(x) = g(y)\} \\ &= (f \times g)^{-1}(\mathit{Eq}(Z)) \end{aligned}$$

in:

$$\mathit{Rel}(F)(\mathit{Pb}(f,g)) = \mathit{Pb}(F(f), F(g)).$$

Proof. (i) By the results from the previous two lemmas:

$$\begin{aligned} \mathrm{Rel}(F)(\mathrm{Ker}(f)) &= \mathrm{Rel}(F)((f \times f)^{-1}(\mathrm{Eq}(Y))) \\ &= (F(f) \times F(f))^{-1}(\mathrm{Rel}(F)(\mathrm{Eq}(Y))) \\ &= (F(f) \times F(f))^{-1}(\mathrm{Eq}(F(Y))) \\ &= \mathrm{Ker}(F(f)). \end{aligned}$$

(ii) Similarly:

$$\begin{aligned} \mathrm{Rel}(F)(\mathrm{Graph}(f)) &= \mathrm{Rel}(F)((f \times \mathrm{id}_Y)^{-1}(\mathrm{Eq}(Y))) \\ &= (F(f) \times \mathrm{id}_{F(Y)})^{-1}(\mathrm{Rel}(F)(\mathrm{Eq}(Y))) \\ &= (F(f) \times \mathrm{id}_{F(Y)})^{-1}(\mathrm{Eq}(F(Y))) \\ &= \mathrm{Graph}(F(f)). \end{aligned}$$

(iii) And:

$$\begin{aligned} \mathrm{Rel}(F)(\mathrm{Im}(\langle f,g\rangle)) &= \mathrm{Rel}(F)(\textstyle\coprod_{f\times g}(\mathrm{Eq}(Z))) \\ &= \textstyle\coprod_{F(f)\times F(g)}(\mathrm{Rel}(F)(\mathrm{Eq}(Z))) \\ &= \textstyle\coprod_{F(f)\times F(g)}(\mathrm{Eq}(F(Z))) \\ &= \mathrm{Im}(\langle F(f), F(g)\rangle) \end{aligned}$$

(iv) Finally:

$$\begin{aligned} \mathrm{Rel}(F)(\mathrm{Pb}(f,g)) &= \mathrm{Rel}(F)((f \times g)^{-1}(\mathrm{Eq}(Z))) \\ &= (F(f) \times F(g))^{-1}(\mathrm{Rel}(F)(\mathrm{Eq}(Z))) \\ &= (F(f) \times F(g))^{-1}(\mathrm{Eq}(F(Z))) \\ &= \mathrm{Pb}(F(f), F(g)). \end{aligned}$$

❑

Once these auxiliary results are in place, the next two propositions establish a series of standard facts about bisimulations and bisimilarity, see [216, Section 5]. We begin with closure properties.

3.2.5. Proposition. *Assume coalgebras* $X \xrightarrow{c} F(X)$, $X' \xrightarrow{c'} F(X')$, *and* $Y \xrightarrow{d} F(Y)$, $Y' \xrightarrow{d'} F(Y')$ *of a polynomial functor* F. *Bisimulations are closed under:*

(i) *reversal: if* $R \subseteq X \times Y$ *is a bisimulation, then so is* $R^{-1} \subseteq Y \times X$.

(ii) *composition: if* $R \subseteq X \times X'$ *and* $S \subseteq X' \times Y$ *are bisimulations, then so is* $(S \circ R) \subseteq X \times Y$.

(iii) *arbitrary unions: if* $R_i \subseteq X \times Y$ *is a bisimulation for each* $i \in I$, *then so is* $\bigcup_{i\in I} R_i \subseteq X \times Y$.

(iv) *inverse images: for homomorphisms* $f\colon X \to Y$ *and* $f'\colon X' \to Y'$, *if* $R \subseteq Y \times Y'$ *is a bisimulation, then so is* $(f \times f')^{-1}(R) \subseteq X \times X'$.

(v) *direct images: for homomorphisms* $f\colon X \to Y$ *and* $f'\colon X' \to Y'$, *if* $R \subseteq X \times X'$ *is a bisimulation, then so is* $\coprod_{f\times f'}(R) \subseteq Y \times Y'$.

Proof. (i) If the relation R is a bisimulation, *i.e.* if $R \subseteq (c \times d)^{-1}(\mathrm{Rel}(F)(R))$, then

$$\begin{aligned} R^{-1} &\subseteq \Big((c \times d)^{-1}(\mathrm{Rel}(F)(R))\Big)^{-1} \\ &= (d \times c)^{-1}\Big(\mathrm{Rel}(F)(R)^{-1}\Big) \\ &= (d \times c)^{-1}(\mathrm{Rel}(F)(R^{-1})) \qquad \text{by Lemma 3.2.1 (ii).} \end{aligned}$$

Notice that the notation $(-)^{-1}$ is used both for inverse images of functions and for inverses of relations.

(ii) Assume $R \subseteq (c \times c')^{-1}(\mathrm{Rel}(F)(R))$ and $S \subseteq (c' \times d)^{-1}(\mathrm{Rel}(F)(S))$. If $(x,y) \in (S \circ R)$, say with $R(x,x')$ and $S(x',y)$, then by assumption $(c(x), c'(x')) \in \mathrm{Rel}(F)(R)$ and $(c'(x'), d(y)) \in \mathrm{Rel}(F)(S)$. Hence $(c(x), d(y)) \in \big(\mathrm{Rel}(F)(S) \circ \mathrm{Rel}(F)(R)\big) = \mathrm{Rel}(F)(S \circ R)$, by Lemma 3.2.1 (iv). We have thus proved the inclusion $(S \circ R) \subseteq (c \times d)^{-1}(\mathrm{Rel}(F)(S \circ R)$, and thus that $S \circ R$ is a bisimulation.

(iii) Assume $R_i \subseteq (c \times d)^{-1}(\mathrm{Rel}(F)(R_i))$, for each $i \in I$. Then

$$\begin{aligned} \textstyle\bigcup_{i\in I} R_i &\subseteq \textstyle\bigcup_{i\in I}(c \times d)^{-1}(\mathrm{Rel}(F)(R_i)) \\ &= (c \times d)^{-1}\big(\textstyle\bigcup_{i\in I}\mathrm{Rel}(F)(R_i)\big) && \text{since inverse image preserves unions} \\ &\subseteq (c \times d)^{-1}\big(\mathrm{Rel}(F)(\textstyle\bigcup_{i\in I} R_i)\big) && \text{by monotony of relation lifting} \\ & && \text{(and of inverse images).} \end{aligned}$$

(iv) If $R \subseteq (d \times d')^{-1}(\mathrm{Rel}(F)(R))$, then:

$$\begin{aligned} &(f \times f')^{-1}(R) \\ &\quad\subseteq (f \times f')^{-1}(d \times d')^{-1}(\mathrm{Rel}(F)(R)) \\ &\quad= (c \times c')^{-1}(F(f) \times F(f'))^{-1}(\mathrm{Rel}(F)(R)) && \text{because } f, f' \text{ are homomorphisms} \\ &\quad= (c \times c')^{-1}(\mathrm{Rel}(F)((f \times f')^{-1}(R))) && \text{by Lemma 3.2.2 (i).} \end{aligned}$$

(v) If $\coprod_{c\times c'}(R) \subseteq \mathrm{Rel}(F)(R)$, then:

$$\begin{aligned} &\textstyle\coprod_{d\times d'}\coprod_{f\times f'}(R) \\ &\quad= \textstyle\coprod_{F(f)\times F(f')}\coprod_{c\times c'}(R) && \text{since } f, f' \text{ are homomorphisms} \\ &\quad\subseteq \textstyle\coprod_{F(f)\times F(f')}(\mathrm{Rel}(F)(R)) && \text{by assumption} \\ &\quad= \mathrm{Rel}(F)(\textstyle\coprod_{f\times f'}(R)) && \text{by Lemma 3.2.2 (ii).} \end{aligned}$$

❑

3.2.6. Proposition. *Let $X \xrightarrow{c} F(X)$, $Y \xrightarrow{d} F(Y)$ and $Z \xrightarrow{e} F(Z)$ be three coalgebras of a polynomial functor F.*

(i) *An arbitrary function $f\colon X \to Y$ is a homomorphism of coalgebras if and only if its graph relation Graph(f) is a bisimulation.*

(ii) *The equality relation Eq(X) on X is a bisimulation equivalence. More generally, for a homomorphism $f\colon X \to Y$, the kernel relation Ker(f) is a bisimulation equivalence.*

(iii) *For two homomorphisms $X \xleftarrow{f} Z \xrightarrow{g} Y$ the image Im$(\langle f, g\rangle)$ is a bisimulation.*

(iv) *For two homomorphisms $X \xrightarrow{f} Z \xleftarrow{g} Y$ in the opposite direction, the pullback relation Pb(f, g) is a bisimulation.*

Proof. By using Lemma 3.2.1.

(i) Because:

$$\begin{aligned}
&\text{Graph}(f) \text{ is a bisimulation}\\
\iff\ &\text{Graph}(f) \subseteq (c \times d)^{-1}(\text{Rel}(F)(\text{Graph}(f)))\\
&\qquad = (c \times d)^{-1}(\text{Graph}(F(f))) \quad \text{by Lemma 3.2.4 (ii)}\\
\iff\ &\forall x, y.\ f(x) = y \Rightarrow F(f)(c(x)) = d(y)\\
\iff\ &\forall x.\ F(f)(c(x)) = d(f(x))\\
\iff\ &f \text{ is a homomorphism of coalgebras from } c \text{ to } d.
\end{aligned}$$

(ii) The fact that equality is a bisimulation follows directly from Lemma 3.2.1 (i). Further, the kernel Ker(f) is a bisimulation because it can be written as Graph$(f) \circ$ Graph$(f)^{-1}$, which is a bisimulation by (i) and by Proposition 3.2.5 (i), (ii).

(iii) We need to prove an inclusion $\text{Im}(\langle f, g\rangle) \subseteq (c \times d)^{-1}(\text{Rel}(F)(\text{Im}(\langle f, g\rangle))) = (c \times d)^{-1}(\text{Im}(\langle F(f), F(g)\rangle))$, using Lemma 3.2.4 (iii). This means that we have to prove: for each $z \in Z$ there is a $w \in F(Z)$ with $c(f(z)) = F(f)(w)$ and $d(f(z)) = F(g)(w)$. But clearly we can take $w = e(z)$.

(iv) What we need is an inclusion $\text{Pb}(f, g) \subseteq (c \times d)^{-1}(\text{Rel}(F)(\text{Pb}(f, g))) = (c \times d)^{-1}(\text{Pb}(F(f), F(g)))$, by Lemma 3.2.4 (iv). But this amounts to: $f(x) = g(y) \Rightarrow F(f)(c(x)) = G(f)(d(y))$, for all $x \in X, y \in Y$. The implication holds because both f and g are homomorphisms. ❑

We can now also establish some elementary properties of bisimilarity.

3.2.7. Proposition. *Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor, with coalgebras $X \xrightarrow{c} F(X)$, $X' \xrightarrow{c'} F(X')$, and $Y \xrightarrow{d} F(Y)$, $Y' \xrightarrow{d'} F(Y')$.*

(i) *The bisimilarity relation ${}_c\!\leftrightarrow_d \subseteq X \times Y$ is a bisimulation; it is the greatest among all bisimulations between c and d.*

(ii) $\left({}_c\!\leftrightarrow_d\right)^{-1} \subseteq {}_d\!\leftrightarrow_c$ *and* ${}_c\!\leftrightarrow_{c'} \circ {}_{c'}\!\leftrightarrow_d \subseteq {}_c\!\leftrightarrow_d$.

(iii) *The bisimilarity relation ${}_c\!\leftrightarrow_c \subseteq X \times X$ for a single coalgebra is a bisimulation equivalence.*

(iv) *For homomorphisms of coalgebras $f\colon X \to Y$ and $f'\colon X' \to Y'$ one has, for $x \in X, x' \in X'$,*

$$f(x)\ {}_d\!\leftrightarrow_{d'} f'(x') \iff x\ {}_c\!\leftrightarrow_{c'} x'$$

Proof. (i) By Proposition 3.2.5 (iii).

(ii) The first inclusion follows from Proposition 3.2.5 (i) and the second one from Proposition 3.2.5 (ii).

(iii) The bisimilarity relation ${}_c\!\leftrightarrow_c \subseteq X \times X$ is reflexive, because the equality relation Eq$(X) \subseteq X \times X$ is a bisimulation, see Proposition 3.2.5 (ii). Symmetry and transitivity of ${}_c\!\leftrightarrow_c$ follow from (ii).

(iv) Since ${}_d\!\leftrightarrow_{d'} \subseteq Y \times Y'$ is a bisimulation, so is $(f \times f')^{-1}({}_d\!\leftrightarrow_{d'}) \subseteq X \times X'$, by Proposition 3.2.5 (iv). Hence $(f \times f')^{-1}({}_d\!\leftrightarrow_{d'}) \subseteq {}_c\!\leftrightarrow_{c'}$, which corresponds to the implication ($\Longrightarrow$).

Similarly we obtain an inclusion $\coprod_{f \times f'}({}_c\!\leftrightarrow_{c'}) \subseteq {}_d\!\leftrightarrow_{d'}$ from Proposition 3.2.5 (v), which yields ($\Longleftarrow$). ❑

Exercises

3.2.1. (i) Prove that relation lifting Rel(F) for a polynomial functor F *without powersets* $\mathcal{P}$ preserves non-empty intersections of relations: for $I \neq \emptyset$,

$$\text{Rel}(F)(\textstyle\bigcap_{i \in I} R_i) \quad = \quad \bigcap_{i \in I} \text{Rel}(F)(R_i)$$

(ii) Assume now that F is not only without powersets $\mathcal{P}$ but also without exponents $(-)^A$—for A infinite, since such exponents for finite A can be obtained via repeated products $\times$. Prove that relation lifting Rel(F) preserves unions of ascending chains of relations: if $S_0 \subseteq S_1 \subseteq S_2 \subseteq \cdots$ then:

$$\text{Rel}(F)(\textstyle\bigcup_{n \in \mathbb{N}} S_n) \quad = \quad \bigcup_{n \in \mathbb{N}} \text{Rel}(F)(R_n).$$

Which additional closure properties hold for bisimulations for coalgebras of such functors?

3.2.2. (i) Check that if $\leq$ is a preorder on a set X, then Rel$(F)(\leq)$ is also a preorder on $F(X)$.

(ii) Prove the same with 'poset' instead of 'preorder', for a polynomial functor F without powerset.

(iii) Prove that a Galois connection $f \dashv g$ in:

$$(X, \leq_X) \underset{g}{\overset{f}{\rightleftarrows}} (Y, \leq_Y)$$

yields a "lifted" Galois connection $F(f) \dashv F(g)$ in:

$$(FX, \text{Rel}(F)(\leq_X)) \underset{F(g)}{\overset{F(f)}{\rightleftarrows}} (FY, \text{Rel}(F)(\leq_Y))$$

3.2.3. Check that, in analogy with Proposition 3.2.5, congruences are closed under inverses, composition, arbitrary intersections, and under inverse and direct images.

3.2.4. Prove the following analogue of Propositions 3.2.6 for algebras $F(X) \xrightarrow{a} X$, $F(Y) \xrightarrow{b} Y$ and $F(Z) \xrightarrow{c} Z$ of a polynomial functor F.

(i) A function $f\colon X \to Y$ is a homomorphism of algebras if and only if its graph relation Graph$(f) \subseteq X \times Y$ is a congruence.

(ii) The kernel relation Ker(f) of an algebra homomorphism $f\colon X \to Y$ is always a congruence equivalence.

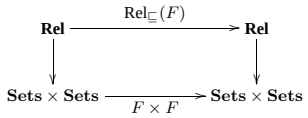
(iii) The image Im$(\langle f, g\rangle)$ of a pair of algebra homomorphisms $X \xleftarrow{f} Z \xrightarrow{g} Y$ is a congruence.

(iv) The pullback relation Pb(f, g) of a span of algebra homomorphisms $X \xrightarrow{f} Z \xleftarrow{g} Y$ is a congruence.

3.2.5. Let $R \subseteq X \times X$ be an arbitrary relation on the state space of a coalgebra, and let $\overline{R}$ be the least equivalence relation containing R. Prove that if R is a bisimulation, then $\overline{R}$ is a bisimulation equivalence.

[*Hint.* Write $\overline{R}$ as union of iterated compositions R^n for $n \in \mathbb{N}$.]

3.2.6. Check that lax relation lifting as introduced in Exercise 3.1.5 forms a functor in:



and that simulations are coalgebras of this functor $\mathrm{Rel}_{\sqsubseteq}(F)$—like in Lemma 3.2.3.

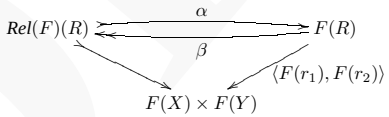
3.2.7. This exercise describes a simple characterisation from [85] of when a function is definable by induction. The analogue for coinduction is in Excercise 4.2.3.
Consider an initial algebra $F(A) \xrightarrow{\cong} A$ of a polynomial functor F, where $A \neq \emptyset$ Prove that a function $f\colon A \to X$ is defined by initiality (*i.e.* is int_a for some algebra $a\colon T(X) \to X$ on its codomain) if and only its kernel $\mathrm{Ker}(f)$ is a congruence.
[*Hint.* Extend the induced map $F(A)/\mathrm{Ker}(F(f)) \to X$ along $F(A)/\mathrm{Ker}(F(f)) \rightarrowtail F(X)$ by using an aribitrary element in $F(A)/\mathrm{Ker}(F(f))$ obtained from $A \neq \emptyset$.]

3.3 Bisimulations as spans

This section continues the investigation of bisimulations, and focusses especifically, on the relation between the definition of bisimulation used here, given in terms of relation lifting, and an earlier definition given by Aczel and Mendler [4, 7]. One of the main results is that these definitions are equivalent, see Theorem 3.3.2.

The first lemma below establishes an important technical relationship which forms the basis for the subsequent theorem. The lemma uses that relations can be considered as sets themselves. From a logical perspective this involves a form of comprehension, see *e.g.* [130, Chapter 4, Section 6] or [117].

3.3.1. Lemma. *Let F be a polynomial functor, and $R \subseteq X \times Y$ be an arbitrary relation, written via explicit functions $\langle r_1, r_2\rangle\colon R \rightarrowtail X \times Y$. The lifted relation $\mathit{Rel}(F)(R)$, considered as a set, is a* **retract** *of $F(R)$: there are functions $\alpha\colon \mathit{Rel}(F)(R) \to F(R)$ and $\beta\colon F(R) \to \mathit{Rel}(F)(R)$ with $\beta \circ \alpha = \mathit{id}_{\mathit{Rel}(F)(R)}$. Moreover, these α and β make the following triangle commute.*



This means that $\mathit{Rel}(F)(R) \rightarrowtail F(X) \times F(Y)$ is the image of $F(R) \to F(X) \times F(Y)$.

Proof. The functions α and β are constructed by induction on the structure of F. In the two base cases where F is the identity functor or a constant functor, α and β are each other's inverses. We shall consider two induction steps, for product and powerset.

If F is a product $F_1 \times F_2$, we may assume appropriate functions $\alpha_i\colon \mathrm{Rel}(F_i)(R) \to F_i(R)$ and $\beta_i\colon F_i(R) \to \mathrm{Rel}(F_i)(R)$, for $i = 1, 2$. The aim is to construct functions α, β in:

$$\left(\begin{array}{r} \{((u_1,u_2),(v_1,v_2)) \,|\, \mathrm{Rel}(F_1)(R)(u_1,v_1) \wedge \\ \mathrm{Rel}(F_2)(R)(u_2,v_2)\} \end{array} \right) \underset{\beta}{\overset{\alpha}{\rightleftarrows}} F_1(R) \times F_2(R)$$

The definitions are obvious:

$$\begin{array}{rcl} \alpha((u_1,u_2),(v_1,v_2)) & = & (\alpha_1(u_1,v_1), \alpha_2(u_2,v_2)) \\ \beta(w_1,w_2) & = & ((\pi_1\beta_1(w_1), \pi_1\beta_2(w_2)), (\pi_2\beta_1(w_1), \pi_2\beta_2(w_2))). \end{array}$$

If F is a powerset $\mathcal{P}(F_1)$, we may assume functions $\alpha_1\colon \mathrm{Rel}(F_1)(R) \to F_1(R)$ and $\beta_1\colon F_1(R) \to \mathrm{Rel}(F_1)(R)$ as in the lemma. We have to construct:

$$\left(\begin{array}{r} \{(U,V) \,|\, \forall u \in U.\, \exists v \in V.\, \mathrm{Rel}(F_1)(R)(u,v) \wedge \\ \forall v \in V.\, \exists u \in U.\, \mathrm{Rel}(F_1)(R)(u,v)\} \end{array} \right) \underset{\beta}{\overset{\alpha}{\rightleftarrows}} \mathcal{P}(F_1(R))$$

In this case we define:

$$\begin{array}{rcl} \alpha(U,V) & = & \{\alpha_1(u,v) \mid u \in U \wedge v \in V \wedge \mathrm{Rel}(F_1)(R)(u,v)\} \\ \beta(W) & = & (\{\pi_1\beta_1(w) \mid w \in W\}, \{\pi_2\beta_1(w) \mid w \in W\}) \end{array}$$

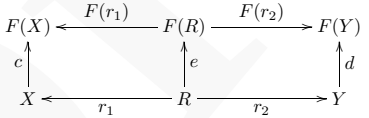
Then, using that $\beta_1 \circ \alpha_1 = \mathrm{id}$ holds by assumption, we compute:

$$\begin{array}{ll} & (\beta \circ \alpha)(U,V) \\ = & (\{\pi_1\beta_1\alpha_1(u,v) \mid u \in U \wedge v \in V \wedge \mathrm{Rel}(F)(R)(u,v)\}, \\ & \{\pi_2\beta_1\alpha_1(u,v) \mid u \in U \wedge v \in V \wedge \mathrm{Rel}(F)(R)(u,v)\}) \\ = & (\{u \in U \mid \exists v \in V.\, \mathrm{Rel}(F_1)(R)(u,v)\}, \{v \in V \mid \exists u \in U.\, \mathrm{Rel}(F_1)(R)(u,v)\}) \\ = & (U,V). \end{array}$$

❑

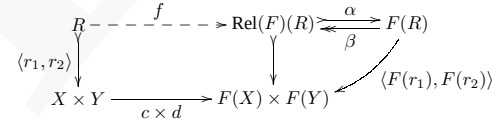
The formulation of bisimulation that we are using here relies on relation lifting, see Definition 3.1.2 and Lemma 3.2.3 (ii), where bisimulations for coalgebras of a functor F are described as $\mathrm{Rel}(F)$-coalgebras. This comes from [117]. An earlier definition was introduced by Aczel and Mendler, see [4, 7]. With the last lemma we can prove the equivalence of these definitions.

3.3.2. Theorem. *Let $X \xrightarrow{c} F(X)$ and $Y \xrightarrow{d} F(Y)$ be two coalgebras of a polynomial functor F. A relation $\langle r_1, r_2\rangle\colon R \rightarrowtail X \times Y$ is a bisimulation for c and d if and only if R is an* **Aczel-Mendler bisimulation***: R itself is the carrier of some coalgebra $e\colon R \to F(R)$, making the the legs r_i homomorphisms of coalgebras, as in:*



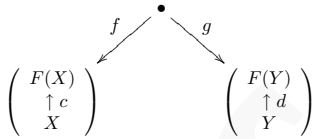
Thus, briefly: R carries a $\mathit{Rel}(F)$-coalgebra if and only if it carries and F-coalgebra making the diagram commute.

Proof. In Lemma 3.2.3 (ii) we already saw that $R \subseteq X \times Y$ is a bisimulation according to Definition 3.1.2 if and only it carries a $\mathrm{Rel}(F)$-coalgebra; that is, if and only if the function $c \times d\colon X \times Y \to F(X) \times F(Y)$ restricts to a necessarily unique function $f\colon R \to \mathrm{Rel}(F)(R)$, making the square on the left below commute.



The functions α, β in the triangle on the right form the retract from the previous lemma. Then, if R is a bisimulation according to Definition 3.1.2, there is a function f as indicated, so that $\alpha \circ f\colon R \to F(R)$ yields an F-coalgebra on R making the legs r_i homomorphisms of coalgebras. Conversely, if there is a coalgebra $e\colon R \to F(R)$ making the r_i homomorphisms, then $\beta \circ e\colon R \to \mathrm{Rel}(F)(R)$ shows that $R \subseteq (c \times d)^{-1}(\mathrm{Rel}(F)(R))$. ❑

3.3.3. Corollary. *Two elements $x \in X$ and $y \in Y$ of coalgebras $X \xrightarrow{c} F(X)$ and $Y \xrightarrow{d} F(Y)$ of a polynomial functor F are bisimilar if and only there is a span of coalgebra homomorphisms:*



with $x = f(z)$ and $y = g(z)$, for some element z.

Proof. If x and y are bisimilar, then they are contained in some bisimulation $R \subseteq X \times Y$. By the previous result, this relation carries a coalgebra structure making the two legs $X \leftarrow R \rightarrow Y$ homomorphisms, and thus a span of coalgebras.

Conversely, if there is a span as indicated in the corollary, then the image $\mathrm{Im}(\langle f, g\rangle) \subseteq X \times Y$ of the tuple of the two functions is a bisimulation by Proposition 3.2.6 (iii). It contains (x, y) by assumption, so that x and y are bisimilar. ❑

We postpone a discussion of the different formulations of the notion of bisimulation to the end of this section. At this stage we shall use the new "Aczel-Mendler" bisimulations in the following standard result—specifically in point (ii)—about monos and epis in categories of coalgebras.

3.3.4. Theorem. *Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor.*

(i) *For a coalgebra $c\colon X \to F(X)$ and a bisimulation equivalence $R \subseteq X \times X$, the quotient set X/R carries a unique coalgebra structure, written as $c/R\colon X/R \to F(X/R)$, making the canonical quotient map $[-]_R\colon X \twoheadrightarrow X/R$ a homomorphism of coalgebras, as in:*

$$\begin{array}{ccc} F(X) & \xrightarrow{F([-]_R)} & F(X/R) \\ {\scriptstyle c}\uparrow & & \uparrow{\scriptstyle c/R} \\ X & \xrightarrow[[-]_R]{} & X/R \end{array}$$

(ii) *A homomorphism of coalgebras $f\colon X \to Y$ from $X \xrightarrow{c} F(X)$ to $Y \xrightarrow{d} F(Y)$ is a monomorphism / epimorphism in the category $\mathbf{CoAlg}(F)$ if and only if f is an injective / surjective function between the underlying sets.*

(iii) *Every homomorphism of coalgebras factors as an epimorphism followed by a monomorphism in $\mathbf{CoAlg}(F)$. This factorisation is essentially unique because of the following "diagonal-fill-in" property. For each commuting square of coalgebra homomorphisms as below, there is a unique diagonal homomorphism making both triangles commute.*

*This means that monomorphisms and epimorphisms in the category $\mathbf{CoAlg}(F)$ form a so-called **factorisation system**, see [30, 16].*

As an aside, the result (ii) that a monomorphism between coalgebras is an injective function holds for polynomial functors, but not for arbitrary functors, as shown in [104].

Proof. (i) It suffices to prove that $F([-]_R) \circ c$ is constant on R. But this is obvious:

$$\begin{array}{rcll} R & \subseteq & (c \times c)^{-1}(\mathrm{Rel}(F)(R)) & \text{since } R \text{ is a bisimulation} \\ & = & (c \times c)^{-1}(\mathrm{Rel}(F)(\mathrm{Ker}([-]_R))) & \\ & = & (c \times c)^{-1}(\mathrm{Ker}(F([-]_R))) & \text{by Lemma 3.2.4 (i)} \\ & = & \mathrm{Ker}(F([-]_R) \circ c). & \end{array}$$

(ii) It is standard that if f is injective / surjective, then it is a monomorphism / epimorphism in $\mathbf{Sets}$—see also Exercise 2.5.6—and hence also in $\mathbf{CoAlg}(F)$. Conversely, assume first that f is a monomorphism in $\mathbf{CoAlg}(F)$. The kernel $\langle r_1, r_2\rangle\colon \mathrm{Ker}(f) \rightarrowtail X \times X$ is a bisimulation by Proposition 3.2.6 (ii). Hence it carries an F-coalgebra structure by the previous theorem, making the r_i homomorphisms. From $f \circ r_1 = f \circ r_2$, we can conclude that $r_1 = r_2$, since f is a monomorphism in $\mathbf{CoAlg}(F)$. But $r_1 = r_2$ yields that f is injective:

$$\begin{array}{rcl} f(x_1) = f(x_2) & \Longrightarrow & (x_1, x_2) \in \mathrm{Ker}(f) \\ & \Longrightarrow & x_1 = r_1(x_1, x_2) = r_2(x_1, x_2) = x_2. \end{array}$$

Next, assume that f is an epimorphism in $\mathbf{CoAlg}(F)$. There is a short categorical argument that tells that f is then an epi in $\mathbf{Sets}$, and thus surjective: the forgetful functor $\mathbf{CoAlg}(F) \to \mathbf{Sets}$ creates colimits, see [216, Proposition 4.7]. But here we shall give an explicit argument. Recall from Exercise 2.1.13 that coproducts of coalgebras exist. Specifically we can form the coproduct of the coalgebra $d\colon Y \to F(Y)$ with itself. It is $[F(\kappa_1) \circ d, F(\kappa_2) \circ d]\colon Y + Y \to F(Y + Y)$, with the coprojections $\kappa_1, \kappa_2\colon Y \to Y + Y$ as homomorphisms. On its carrier $Y + Y$ we consider the following union of bisimulations, see Proposition 3.2.6.

$$R \stackrel{\text{def}}{=} \mathrm{Eq}(Y + Y) \cup \mathrm{Im}(\langle \kappa_1 \circ f, \kappa_2 \circ f\rangle) \cup \mathrm{Im}(\langle \kappa_2 \circ f, \kappa_1 \circ f\rangle).$$

It is not hard to see that R is an equivalence relation, and thus a bisimulation equivalence. By (i) there is then a unique coalgebra structure on the quotient $(Y + Y)/R$ making the quotient map $[-]_R\colon (Y + Y) \twoheadrightarrow (Y + Y)/R$ a homomorphism. By construction of R we have $[-]_R \circ \kappa_1 \circ f = [-]_R \circ \kappa_2 \circ f$. Since f is assumed to be an epimorphism, this gives us $[-]_R \circ \kappa_1 = [-]_R \circ \kappa_2$. For and arbitrary $y \in Y$ we get $(\kappa_1(y), \kappa_2(y)) \in \mathrm{Ker}([-]_R) = R$. But then $(\kappa_1(y), \kappa_2(y)) \in \mathrm{Im}(\langle \kappa_1 \circ f, \kappa_2 \circ f\rangle) = \{(\kappa_1(f(x)), \kappa_2(f(x))) \mid x \in X\}$. This means that $y = f(x)$ for some $x \in X$. Hence f is surjective.

(iii) Given a homomorphism of coalgebras $f\colon X \to Y$, we know by Proposition 3.2.6 (ii) that the kernel $\mathrm{Ker}(f)$ is a bisimulation equivalence. Hence the quotient $X/\mathrm{Ker}(f)$ carries a coalgebra structure, and yields a standard factorisation:

$$\Big(X \xrightarrow{\ f\ } Y\Big) = \Big(X \twoheadrightarrow X/\mathrm{Ker}(f) \rightarrowtail Y\Big)$$

The map $X \twoheadrightarrow X/\mathrm{Ker}(f)$ is by definition of the coalgebra structure on $X/\mathrm{Ker}(f)$—see point (i)—a homomorphism of coalgebras. Using that this map is an epimorphism yields that $X/\mathrm{Ker}(f) \rightarrowtail Y$ is also a homomorphism of coalgebras. Hence we have a factorisation in the category $\mathbf{CoAlg}(F)$.

Regarding the diagonal-fill-in property, the diagonal is defined via the surjectivity of the top arrow. Hence this diagonal is a homomorphism of coalgebras. ❑

3.3.1 Comparing definitions of bisimulation

We started this chapter by introducing bisimulations via the logical technique of relation lifting, and later showed equivalence to quite different formulations in terms of spans (Theorem 3.3.2 and Corollary 3.3.3). We shall discuss some differences between these "logical" and "span-based" approaches.

1. The logical approach describes bisimulation as relations with a special *property*, whereas the span-based approach uses a certain (coalgebra) *structure* on relations. This aspect of the logical approach is more appropriate, because it is in line with the idea that bisimulations are special kinds of relations. If, in the Aczel-Mendler approach, the coalgebra structure is necessarily unique, existence of this structure also becomes a property. But uniqueness is neither required nor guaranteed. This is slightly unsatisfactory.

2. Relation lifting has been defined by induction on the structure of polynomial functors. Therefore, the logical approach only applies to such a limited collection of functors. The span-based approach applies to much more general functors $F\colon \mathbb{C} \to \mathbb{C}$. This is a big advantage. In practice one does not use all of these functors, but only the (still very general) class of functors preserving so-called weak pullbacks, see the definition below.

3. Relation lifting is a logical technique which is not restricted to the standard classical logic of sets, but may be defined for more general (categorical) logics in terms of "indexed" or "fibred" preorders, see [117, 130]. For instance, one may wish to consider topological spaces with different logics, for instance with predicates given by the subsets which are closed, or open, or both (clopen). Each of those preorders of predicates has different algebraic / logical properties. Thus, the logical approach is more general (or flexible) in another, logical dimension.

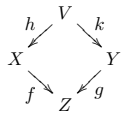
There is no clear answer as to which approach is "the best". We have chosen to start from the logical approach because we consider it to be more intuitive and easier to use in concrete examples. However, from now on we shall freely switch between the two approaches, and use whatever is most convenient in a particular situation.

In the next chapter on invariants we shall encounter the same situation. There is a logical definition based on "predicate lifting". It leads to a notion of invariant, which, in the classical logic of sets, is equivalent to the notion of subcoalgebra, see Theorem 4.2.5. The latter is again defined in terms of structure, and applies to more general functors.

A key result about bisimulations is Theorem 3.4.1 in the next section. It states that two states are bisimilar if and only if they have the same behaviour (*i.e.* are mapped to the same element of the final coalgebra). We shall prove this result for (finite) polynomial functors. In the span-based approach to bisimulation one obtains this result [233] via a restriction to so-called "weak pullback preserving" functors, see Exercise 3.4.4. We shall briefly elaborate on such functors because they play an important rôle in the theory of coalgebras. It involves the notion of (weak) pullback, which is a particular kind of structure (actually limit) in a category.

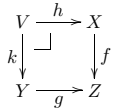
3.3.5. Definition. Let $\mathbb{C}$ be an arbitrary category.

(i) For two morphisms $X \xrightarrow{f} Z \xleftarrow{g} Y$ with a common codomain, a **pullback** is a commuting square of the form



which is universal in the sense that for an arbitrary span $X \xleftarrow{h'} V' \xrightarrow{k'} Y$ with $f \circ h' = g \circ k'$ there is a unique morphism $\ell\colon V' \to V$ with $h \circ \ell = h'$ and $k \circ \ell = k'$. In diagrams

a pullback is often indicated via a small angle, like in:



A **weak pullback** is like a pullback except that only existence and not unique existence of ℓ is required.

(ii) A functor $F\colon \mathbb{C} \to \mathbb{C}$ is called (**weak**) **pullback preserving** if it maps (weak) pullback squares to (weak) pullback squares.

The functors we have been using so far are all weak pullback preserving—which follows by induction.

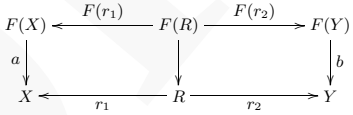
3.3.6. Lemma. *Every polynomial functor* $\mathbf{Sets} \to \mathbf{Sets}$ *preserves weak pullbacks.* ❑

Exercise 3.3.9 below relates the preservation of weak pullbacks to the existence of a "relator" given by relation lifting (in generalised form). This makes it a very natural requirement.

Exercises

3.3.1. Assume a homomorphism of coalgebras $f\colon X \to Y$ has two factorisations $X \twoheadrightarrow U \rightarrowtail Y$ and $X \twoheadrightarrow V \rightarrowtail Y$. Prove that the diagonal-fi ll-in property of Theorem 3.3.4 (iv) yields a unique isomorphism $U \xrightarrow{\cong} V$ commuting with the mono- and epi-morphisms and with the coalgebra structures.

3.3.2. Use Lemma 3.3.1 to prove the following analogue of Theorem 3.3.2. A relation $R \subseteq X \times Y$ is a congruence relation w.r.t. two algebras $a\colon F(X) \to X$ and $b\colon F(Y) \to Y$ of a polynomial functor F if and only it carries an algebra structure $F(R) \to R$ itself, making the two legs $\langle r_1, r_2\rangle\colon R \rightarrowtail X \times Y$ of the inclusion homomorphisms of algebras, as in:



3.3.3. Use the previous exercise to prove the binary induction proof principle in Theorem 3.1.4: every congruence on an initial algebra is reflexive.

3.3.4. Let $R \subseteq X \times X$ be a congruence equivalence relation (*i.e.* both a congruence and an equivalence relation) on an algebra $F(X) \to X$. Prove that the quotient X/R carries an algebra structure $F(X/R) \to X/R$ making the quotient map $X \to X/R$ a homomorphism.
[*Hint.* First define a map $F(X)/\mathrm{Rel}(F)(R) \to X/R$, and then show that the canonical map $F(X)/\mathrm{Rel}(F)(R) \to F(X/R)$ is an isomorphism, via a choice map $X/R \to X$.]

3.3.5. (i) Check that the pullback of two maps $X \xrightarrow{f} Z \xleftarrow{g} Y$ in the category of sets can be described by the set:

$$\mathrm{Pb}(f,g) \quad = \quad \{(x,y) \in X \times Y \mid f(x) = g(y)\},$$

with obvious projections to X and Y.
(ii) Describe the inverse images $h^{-1}(P)$ and $(h \times k)^{-1}(R)$ for predicates P and relations R as pullbacks.
(iii) Similarly, describe the graph $\mathrm{Graph}(f)$ of a function as a pullback of f agains the identity function.

(iv) Prove, using the Axiom of Choice, that a diagram

is a *weak* pullback iff $\mathrm{Pb}(f,g) \subseteq \mathrm{Im}(\langle h,k\rangle)$

3.3.6. Assume $f\colon X \to Y$ is an epimorphism in a category $\mathbf{CoAlg}(F)$, for a polynomial functor F on $\mathbf{Sets}$. Prove that f is the coequaliser in $\mathbf{CoAlg}(F)$ of its own kernel pair $p_1, p_2\colon \mathrm{Ker}(f) \to X$.

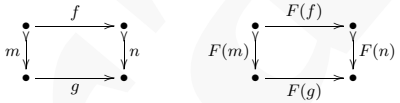
[*Hint.* Use that this property holds in $\mathbf{Sets}$, and lift it to $\mathbf{CoAlg}(F)$.]

3.3.7. Notice that an arbitrary map m in a category $\mathbb{C}$ is a monomorphism if and only if the square

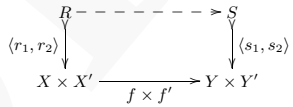
is a (weak) pullback. Let F be a weak pullback preserving functor $\mathbb{C} \to \mathbb{C}$.

(i) Show that F preserves monomorphisms: if m is mono, then so is $F(m)$.

(ii) Prove also that F preserves (actual) pullbacks of monos: if the diagram below on the left is a pullback, then so is the one on the right.



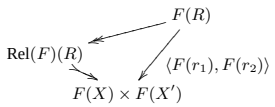
3.3.8. For an arbitrary category $\mathbb{C}$, let $\mathrm{Rel}(\mathbb{C})$ be the category of relations in $\mathbb{C}$. Its objects are monomorphisms $\langle r_1, r_2\rangle\colon R \rightarrowtail X \times X'$. And its morphisms from $\langle r_1, r_2\rangle\colon R \rightarrowtail X \times X'$ to $\langle s_1, s_2\rangle\colon S \rightarrowtail Y \times Y'$ are pairs of morphisms $f\colon X \to Y$, $f'\colon X' \to Y'$ in $\mathbb{C}$ for which there is a necessarily unique morphism $R \to S$ making the diagram below commute.



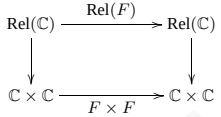
For $R \rightarrowtail X \times Y$ and $R' \rightarrowtail X \times Y$ we write $R \leq R'$ if there is a necessarily unique map $R \to R'$ commuting with the monos. Really, we should use equivalence classes of monos, or subobjects, induced by $\leq$ instead of actual monos.

(i) Check that this yields a category, which comes equipped with a forgetful functor $\mathrm{Rel}(\mathbb{C}) \to \mathbb{C} \times \mathbb{C}$. Check that $\mathrm{Rel}(\mathbf{Sets})$ is the category **Rel** from Lemma 3.2.3.

(ii) Now assume that every arrow in $\mathbb{C}$ can be factored as an epimorphism followed by a monomorphism, and that diagonal-fill-in property from Theorem 3.3.4 (iv) holds. For a mono $m\colon P \rightarrowtail X$ and a map $f\colon X \to Y$ we shall write $\coprod_f(P) \rightarrowtail Y$ for the mono-part of the composite $f \circ m$.
For a functor $F\colon \mathbb{C} \to \mathbb{C}$ define a lifting $\mathrm{Rel}(F)\colon \mathrm{Rel}(\mathbb{C}) \to \mathrm{Rel}(\mathbb{C})$ by mapping a relation $\langle r_1, r_2\rangle\colon R \rightarrowtail X \times X'$ to the mono part on the left of the factorisation:



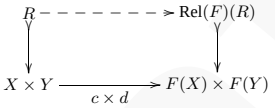
Show how to define $\mathrm{Rel}(F)$ on morphisms in $\mathrm{Rel}(\mathbb{C})$, so that one gets a commuting square:



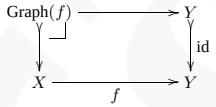
Check that for a polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$ this construction corresponds to relation lifting—so that using the same notation is justified.

(iii) Prove that the functor $\mathrm{Rel}(F)$ as described in (ii) preserves equality relations and reversals of relations.

(iv) Using this general construction one can define a bisimulation as an $\mathrm{Rel}(F)$-coalgebra. It thus consists of a relation $R \rightarrowtail X \times X'$ with a pair of morphisms (coalgebras) $c\colon X \to F(X)$, $d\colon Y \to F(Y)$ such that:

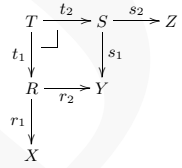


Assume now that $\mathbb{C}$ has pullbacks. Prove that if F preserves weak pullbacks, then $\mathrm{Rel}(F)$ preserves graphs $\mathrm{Graph}(f) \rightarrowtail X \times Y$ of morphisms $f\colon X \to Y$, defined by the pullback:

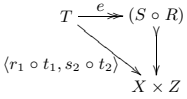


Preservations of graphs means $\mathrm{Rel}(F)(\mathrm{Graph}(f)) = \mathrm{Graph}(F(f))$, of course.

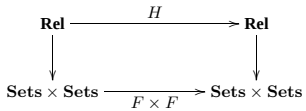
(v) Assume that the "categorical" Axiom of Choice holds in the form that epimorphisms in $\mathbb{C}$ split: for each epi $e\colon X \twoheadrightarrow Y$ there is a "splitting" $s_e\colon Y \to X$ with $e \circ s_e = \mathrm{id}_Y$. Note that such split epis are preserved under functor application.
Prove then that the mapping $\mathrm{Rel}(F)$ preserves composition of relations $\mathrm{Rel}(F)(S \circ R) = \mathrm{Rel}(F)(S) \circ \mathrm{Rel}(R)()$, where the composition of relations $\langle r_1, r_2\rangle\colon R \rightarrowtail X \times Y$ and $\langle s_1, s_2\rangle\colon S \rightarrowtail Y \times Z$ is defined as follows. First form the object T by pullback in:



and take the image:



3.3.9. Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be an arbitrary functor. We write $\mathbf{Rel} = \mathrm{Rel}(\mathbf{Sets})$ as in Lemma 3.2.3 and in the previous exercise. Call a functor $H\colon \mathbf{Rel} \to \mathbf{Rel}$ an F**-relator** if it makes the following diagram commute



and satisfies:

(i) H preserves equality relations, reversals and compositions of relations;
(ii) H preserves graphs in the sense that $H(\mathrm{Graph}(f)) = \mathrm{Graph}(F(f))$.
Prove that F has an F-relator if and only if F preserves weak pullbacks, and that the relator is in that case uniquely determined as $\mathrm{Rel}(F)$—defined like in the previous exercise.
[*Hint.* Use that for a relation $\langle r_1, r_2\rangle \colon R \rightarrowtail X \times Y$ one can write $R = \mathrm{Graph}(r_2) \circ \mathrm{Graph}(r_1)^{-1}$. Similarly, $\mathrm{Pb}(f,g) = \mathrm{Graph}(g)^{-1} \circ \mathrm{Graph}(f)$ and $\mathrm{Im}(\langle h, k\rangle) = \mathrm{Graph}(k) \circ \mathrm{Graph}(h)^{-1}$.]

[The essence of this result comes from [50]. There is some disagreement about the precise definition of an F-relator, see for instance [230, 214], but the most reasonable requirements seem to be precisely those that yield the above equivalence with preservation of weak pullbacks by F. Often these relators are defined with respect to the category **REL** with sets as objects and relations as morphisms, see Example 1.4.2 (iv). But the category **Rel** with relations as objects (that we use) seems more natural in this context, for instance, because it contains bisimulations as coalgebras.]

3.4 Bisimulations and the coinduction proof principle

We have already seen that states of final coalgebras coincide with behaviours, and that bisimilarity is observational indistinguishability. Hence the following fundamental result does not come as a surprise: states are bisimilar if and only if they have the same behaviour, *i.e.* become equal when mapped to the final coalgebra.

3.4.1. Theorem ([233, 216]). *Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be a (finite) polynomial functor which has a final coalgebra $\zeta\colon Z \xrightarrow{\cong} F(Z)$. Let $c\colon X \to F(X)$ and $d\colon Y \to F(Y)$ be arbitrary coalgebras, with associated homomorphisms $\mathsf{beh}_c\colon X \to Z$ and $\mathsf{beh}_d\colon Y \to Z$ given by finality. Two states $x \in X$ and $y \in Y$ are then bisimilar if and only if they have the same behaviour:*

$$x \;{}_c\!\leftrightarrows_d\; y \iff \mathsf{beh}_c(x) = \mathsf{beh}_d(y).$$

In particular, bisimilarity ${}_\zeta\!\leftrightarrows_\zeta \subseteq Z \times Z$ on the final coalgebra is equality.

Proof. ($\Leftarrow$) This is easy since we know by Proposition 3.2.6 (iv) that the pullback relation $\mathrm{Pb}(\mathsf{beh}_c, \mathsf{beh}_d) = \{(x,y) \mid \mathsf{beh}_c(x) = \mathsf{beh}_d(y)\} \subseteq X \times Y$ of two homomorphisms is a bisimulation. Hence it is included in the greatest bisimulation ${}_c\!\leftrightarrows_d$.

($\Rightarrow$) The bisimilarity relation ${}_c\!\leftrightarrows_d$ is itself a bisimulation, so it carries by Theorem 3.3.2 a coalgebra structure $e\colon ({}_c\!\leftrightarrows_d) \to F({}_c\!\leftrightarrows_d)$ making the two legs r_i of the relation $\langle r_1, r_2\rangle : ({}_c\!\leftrightarrows_d) \rightarrowtail X \times Y$ homomorphisms. By finality we then get $\mathsf{beh}_c \circ r_1 = \mathsf{beh}_d \circ r_2$, yielding the required result. ❑

This result gives rise to an important proof method for establishing that two states have the same behaviour. This is method is often referred to as the coinduction proof principle, and goes back to [179]. In corresponds to the uniqueness part of the unique existence property of behaviour maps in Definition 2.3.1.

3.4.2. Corollary (Coinduction proof principle). *Two states have the same behaviour if and only if there is a bisimulation that contains them.*

Consequently: every bisimulation on a final coalgebra is contained in the equality relation. ❑

The second formulation in this corollary is the dual of the binary induction principle from Theorem 3.1.4.

As we shall see in Example 3.4.5 below, it may sometimes require a bit of ingenuity to produce an appropriate bisimulation. The standard way to find such a bisimulation is to start with the given equation as relation, and close it off with successor states until no new elements appear. In that case one has only "circularities". Formalising this approach lead to what is sometimes called circular rewriting, see *e.g.* [91].

3.4.3. Corollary. *Call a coalgebra $c\colon X \to F(X)$ **observable** if its bisimilarity relation ${}_c\!\leftrightarrows_c$ is equality on X. This is equivalent to a generalisation of the formulation used in Exercise 2.5.11 for deterministic automata: the associated behaviour map $\mathsf{beh}_c\colon X \to Z$ to the final coalgebra $Z \xrightarrow{\cong} F(Z)$, if any, is injective.* ❑

Observable coalgebras are called *simple* in [216]. Coalgebras can always be forced to be observable via quotienting, see Exercise 3.4.1 below.

Bisimilarity is closely related to equivalence of automata, expressed in terms of equality of accepted languages (see Corollary 2.3.6). This result, going back to [189], will be illustrated next.

3.4.4. Corollary. *Consider two deterministic automata $\langle \delta_i, \epsilon_i\rangle\colon S_i \to S_i^A \times \{0,1\}$ with initial states $s_i \in S_i$, for $i = 1, 2$. These states s_1, s_2 are called **equivalent** if they accept the same language. The states s_1 and s_2 are then equivalent if and only if they are bisimilar.*

Proof. Because the accepted languages are given by the behaviours $\mathsf{beh}_{\langle \delta_i, \epsilon_i\rangle}(s_i) \in \mathcal{P}(A^\star)$ of the initial states, see Corollary 2.3.6 (ii). ❑

Early on in Section 1.2 we already saw examples of coinductive reasoning for sequences. Here we continue to illustrate coinduction with languages.

3.4.5. Example (Equality of regular languages [213]). In Corollary 2.3.6 (ii) we have seen that the set $\mathcal{L}(A) = \mathcal{P}(A^\star)$ of languages over an alphabet A forms a final coalgebra, namely for the deterministic automaton functor $X \mapsto X^A \times \{0,1\}$. We recall that the relevant coalgebra structure on $\mathcal{L}(A)$ is given on a language $L \subseteq A^\star$ by:

$$\begin{array}{ll} L \xrightarrow{a} L_a & \text{where } L_a = \{\sigma \in A^\star \mid a \cdot \sigma \in L\} \text{ is the } a\text{-derivative of } L \\ L\!\downarrow 1 \iff \langle\rangle \in L & \text{which may simply be written as } L\!\downarrow. \end{array}$$

The subset of so-called **regular languages** is built up inductively from constants

$$0 = \emptyset, \qquad 1 = \{\langle\rangle\}, \qquad \{a\}, \text{ for } a \in A, \text{ usually written simply as } a$$

and the three operations of **union**, **concatenation** and **Kleene star**:

$$\begin{array}{rcl} K + L & = & K \cup L \\ KL & = & \{\sigma \cdot \tau \mid \sigma \in K \wedge \tau \in L\} \\ K^* & = & \bigcup_{n \in \mathbb{N}} K^n, \quad \text{where } K^0 = 1 \text{ and } K^{n+1} = KK^n. \end{array}$$

See also Excercise 3.4.5 below. For example, the regular language $a(a+b)^*b$ consists of all (finite) words consisting of letters a, b only, that start with an a and end with a b. Regular languages can be introduced in various other ways, for example as the languages accepted by deterministic and non-deterministic automata with a finite state space (via what is called Kleene's theorem [158]), or as the languages generated by regular grammars. Regular languages (or expressions) are used in many situations, such lexical analysis (as patterns for tokens), or text editing and retrieval (for context searches). Regular expressions, see Exercise 3.4.5, are often used as search strings in a Unix/Linux environement; for example in the command `grep`, for "general regular expression parser".

An important topic is proving equality of regular languages. There are several approaches, namely via unpacking the definitions, via algebraic reasoning using a complete set of laws (see [159] and also [138]), or via minimalisation of associated automata. A fourth, coinductive approach is introduced in [213] using bisimulations. It is convenient, and will be illustrated here.

Recall from Section 3.1 that a relation $R \subseteq \mathcal{L}(A) \times \mathcal{L}(A)$ is a bisimulation if for all languages L, K,

$$R(L,K) \implies \begin{cases} R(L_a, K_a) & \text{for all } a \in A \\ L\!\downarrow \text{ iff } K\!\downarrow \end{cases}$$

The coinduction proof principle then says:

$$L = K \iff \text{there is a bisimulation } R \subseteq \mathcal{L}(A) \times \mathcal{L}(A) \text{ with } R(L,K). \quad (3.1)$$

In order to use this principle effectively the following rules for derivatives L_a and termination $L\!\downarrow$ are useful.

$$\begin{array}{rcll} 0_a & = & 0 & \neg(0\!\downarrow) \\ 1_a & = & 0 & 1\!\downarrow \\ b_a & = & \begin{cases} 1 & \text{if } b = a \\ 0 & \text{otherwise} \end{cases} & \neg(b\!\downarrow) \\ (K+L)_a & = & K_a + L_a & K+L\!\downarrow \text{ iff } K\!\downarrow \text{ or } L\!\downarrow \\ (KL)_a & = & \begin{cases} K_a L + L_a & \text{if } K\!\downarrow \\ K_a L & \text{otherwise} \end{cases} & KL\!\downarrow \text{ iff } K\!\downarrow \wedge L\!\downarrow \\ \big(K^*\big)_a & = & K_a K^* & K^*\!\downarrow. \end{array}$$

We shall illustrate the use of the coindution proof principle (3.1) for establishing equality of regular languages via two examples.

1. For an arbitrary element a in the alphabet A one has

$$(1+a)^* \;=\; a^*.$$

As candidate bisimulation R in (3.1) we take:

$$R \;=\; \{((1+a)^*, a^*)\} \cup \{(0,0)\}.$$

The termination requirement obviously holds, so we concentrate on derivatives. First, for a itself:

$$\big((1+a)^*\big)_a = (1+a)_a\,(1+a)^* = (1_a + a_a)\,(1+a)^* = (0+1)\,(1+a)^* = (1+a)^*$$

and similarly:

$$\big(a^*\big)_a = a_a\,a^* = 1\,a^* = a^*.$$

Hence the pair of a-derivatives $(\,((1+a)^*)_a, (a^*)_a\,) = ((1+a)^*, a^*)$ is again in the relation R. Similarly, $(0_a, 0_a) = (0,0) \in R$. And for an element $b \neq a$ we similarly have $(\big((1+a)^*\big)_b, \big(a^*\big)_b) = (0,0) \in R$. This shows that R is a bisimulation, and completes the proof. The reader may wish to compare it to an alternative proof using the definition of Kleene star $(-)^*$.

2. Next we restrict ourselves to an alphabet $A = \{a, b\}$ consisting of two (different) letters only. Consider the two languages

$$\begin{array}{rcl} E(b) & = & \{\sigma \in A^\star \mid \sigma \text{ contains an even number of } b\text{'s}\} \\ O(b) & = & \{\sigma \in A^\star \mid \sigma \text{ contains an odd number of } b\text{'s}\}. \end{array}$$

(We consider $0 \in \mathbb{N}$ to be even.) Using the definitions of derivative and termination, it is not hard to see that:

$$\begin{array}{rclcrcl} E(b)_a & = & E(b) & \qquad & E(b)_b & = & O(b) \qquad E(b)\!\downarrow \\ O(b)_a & = & O(b) & & O(b)_b & = & E(b) \qquad \neg\, O(b)\!\downarrow. \end{array}$$

Our aim is to prove the equality:

$$E(b) \;=\; a^* + a^* b (a + ba^* b)^* ba^*$$

via coinduction. This requires by (3.1) a bisimulation R containing both sides of the equation. We take:

$$\begin{array}{rcl} R & = & \{(E(b), K)\} \cup \{(O(b), L)\} \text{ where} \\ & & K \;=\; a^* + a^* b (a + ba^* b)^* ba^* \quad \text{(the right-hand side of the equation)} \\ & & L \;=\; (a + ba^* b)^* ba^*. \end{array}$$

The computations that show that R is a bisimulation use the above computation rules for derivatives plus some obvious properties of the regular operations (like $X + 0 = X$ and $1X = X$):

$$\begin{array}{rcl} K_a & = & \big(a^*\big)_a + \big(a^*\big)_a b(a+ba^*b)^*ba^* + \big(b(a+ba^*b)^*ba^*\big)_a \\ & = & a^* + a^*b(a+ba^*b)^*ba^* + b_a(a+ba^*b)^*ba^* \\ & = & K + 0(a+ba^*b)^*ba^* \\ & = & K \\ K_b & = & \big(a^*\big)_b + \big(a^*\big)_b b(a+ba^*b)^*ba^* + \big(b(a+ba^*b)^*ba^*\big)_b \\ & = & 0 + 0b(a+ba^*b)^*ba^* + b_b(a+ba^*b)^*ba^* \\ & = & 0 + 0 + 1(a+ba^*b)^*ba^* \\ & = & L \\ L_a & = & \big((a+ba^*b)^*\big)_a ba^* + \big(ba^*\big)_a \\ & = & \big(a+ba^*b\big)_a (a+ba^*b)^*ba^* + b_a a^* \\ & = & (a_a + b_a a^* b)(a+ba^*b)^*ba^* + 0a^* \\ & = & (1 + 0a^*b)(a+ba^*b)^*ba^* + 0 \\ & = & L \\ L_b & = & \big((a+ba^*b)^*\big)_b ba^* + \big(ba^*\big)_b \\ & = & \big(a+ba^*b\big)_b (a+ba^*b)^*ba^* + b_b a^* \\ & = & (a_b + b_b a^* b)(a+ba^*b)^*ba^* + 1a^* \\ & = & a^*b(a+ba^*b)^*ba^* + a^* \\ & = & K. \end{array}$$

This shows that $(U, V) \in R$ implies both $(U_a, V_a) \in R$ and $(U_b, V_b) \in R$.

Further:

$$\begin{array}{rcl} K\!\downarrow & \iff & a^*\!\downarrow \text{ or } a^*b(a+ba^*b)^*ba^*\!\downarrow \\ & \iff & \text{true} \\ L\!\downarrow & \iff & (a+ba^*b)^*\!\downarrow \;\wedge\; ba^*\!\downarrow \\ & \iff & \text{true} \;\wedge\; b\!\downarrow \;\wedge\; a^*\!\downarrow \\ & \iff & \text{true} \;\wedge\; \text{false} \;\wedge\; \text{true} \\ & \iff & \text{false}. \end{array}$$

This shows that R is a bisimulation. As a result we obtain $E(b) = K$, as required, but also, $O(b) = L$.

This concludes the example. For more information, see [213, 220, 215].

There are many more examples of coinductive reasoning in the literature, in various areas: non-well-founded sets [4, 35], processes [180], functional programs [96], streams [215, 113] (with analytic functions as special case [192]), datatypes [112], domains [73, 195], *etc.*

Exercises

3.4.1. Check that for an arbitrary coalgebra $c\colon X \to F(X)$ of a polynomial functor, the induced coalgebra $c/\!\leftrightarrows : X/\!\leftrightarrows \to F(X/\!\leftrightarrows)$ is observable—using Theorem 3.3.4 (i) and Proposition 3.2.7 (iii). Is the mapping $c \mapsto c/\!\leftrightarrows$ functorial?
Note that since the canonical map $[-]\colon X \twoheadrightarrow X/\!\leftrightarrows$ is a homomorphism, its graph is a bisimulation. Hence a state $x \in X$ is bisimilar to its equivalence class $[x] \in X/\!\leftrightarrows$. This means that making a coalgebra observable does not change the behaviour.

3.4.2. (i) ([216, Theorem 8.1]) Prove that a coalgebra c is observable (or simple) if and only if it has no proper quotients: every epimorphism $c \twoheadrightarrow d$ is an isomorphism.
[*Hint.* Consider the kernel of such a map.]
(ii) Conclude that there is at most one homormophism to an observable coalgebra.

3.4.3. Prove the following analogue of Theorem 3.4.1 for algebras $a\colon F(X) \to X$ and $b\colon F(Y) \to Y$ of a polynomial functor F, with an initial algebra $F(A) \xrightarrow{\cong} A$. Two elements $x \in X$ and $y \in Y$ are interpretations $x = \mathsf{int}_a(t)$ and $y = \mathsf{int}_b(t)$ of the same element $t \in A$ if and only if the pair (x, y) is in each congruence relation $R \subseteq X \times Y$.
Conclude that for coalgebras c, d and algebras a, b:

$$\begin{array}{rcl} \mathrm{Pb}(\mathsf{beh}_c, \mathsf{beh}_d) & \stackrel{\text{def}}{=} & (\mathsf{beh}_c \times \mathsf{beh}_d)^{-1}(\mathrm{Eq}) \\ & = & \bigcup\{R \mid R \text{ is a bisimulation on the carriers of } c, d\} \\ \mathrm{Im}(\langle \mathsf{int}_a, \mathsf{int}_b\rangle) & \stackrel{\text{def}}{=} & \coprod_{\mathsf{int}_a \times \mathsf{int}_b}(\mathrm{Eq}) \\ & = & \bigcap\{R \mid R \text{ is a congruence on the carriers of } a, b\}. \end{array}$$

3.4.4. Prove Theorem 3.4.1 for weak pullback preserving functors F with a final coalgebra (like in [233]) by showing that for coalgebras $X \xrightarrow{c} FX$ and $Y \xrightarrow{d} FY$:
(i) each Aczel-Mendler bisimulation $R \subseteq X \times Y$ satisfies $R \subseteq \mathrm{Pb}(\mathsf{beh}_c, \mathsf{beh}_d)$.
(ii) the relation $\mathrm{Pb}(\mathsf{beh}_c, \mathsf{beh}_d) \subseteq X \times Y$ is an Aczel-Mendler bisimulation.

3.4.5. Fix an alphabet A and consider the polynomial functor

$$\mathcal{R}(X) \quad = \quad 1 + 1 + A + (X \times X) + (X \times X) + X.$$

(i) Show that the initial algebra RE of $\mathcal{R}$ is the set of **regular expressions**, given by the BNF syntax:

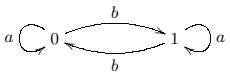
$$E \quad ::= \quad 0 \mid 1 \mid a \mid E + E \mid EE \mid E^*$$

where $a \in A$.
(ii) Define an interpretation map $\mathsf{int}\colon RE \to \mathcal{P}(A^*) = \mathcal{L}(A)$ by initiality, whose image contains precisely the regular languages.

3.4.6. (From [213]) Prove the following equality of regular languages (over the alphabet $\{a, b\}$) by coinduction.

$$((b^*a)^*ab^*)^* \quad = \quad 1 + a(a+b)^* + (a+b)^*aa(a+b)^*.$$

3.4.7. Prove that the language $K = a^* + a^*b(a + ba^*b)^*ba^*$ of words with an even numbers of b's from Example 3.4.5 is the language that is accepted by the following finite automaton:



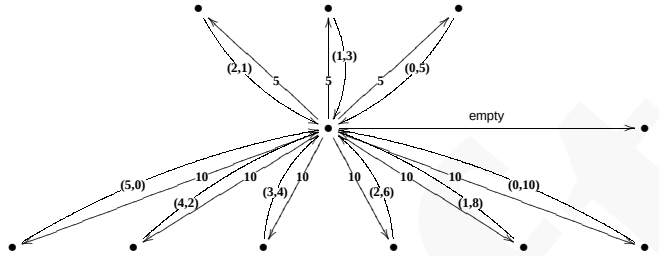


Figure 3.1: Transition diagram of a machine for changing 5 and 10 Euro notes into coins.

with 1 both as initial and as final state. More formally, this automaton is $\langle \delta, \epsilon\rangle\colon \{0,1\} \to \{0,1\}^{\{a,b\}} \times \{0,1\}$ with

$$\begin{array}{lllllllll} \delta(0)(a) & = & 0 & \quad \delta(0)(b) & = & 1 & \quad \epsilon(0) & = & 0 \\ \delta(1)(a) & = & 1 & \quad \delta(1)(b) & = & 0 & \quad \epsilon(1) & = & 1. \end{array}$$

3.5 Process semantics

This section will introduce a semantics for processes using the final coalgebra of the finite powerset functor $\mathcal{P}_{\mathrm{fin}}$. It captures the behaviour of so-called finitely branching transition systems. This forms an illustration of many of the ideas we have seen so far, like behavioural interpretations via finality and compositional interpretations via initiality. Also, we shall see how the coalgebraic notion of bisimilarity froms a congruence—an algebraic notion. The material in this section builds on [233, 222], going back to [212]. It will be put in a broader context via distributive laws in Chapter 6.

A first, non-trivial question is: what is a process? Usually one understands it as a running program. Thus, a sequential program, transforming input to output, when in operation forms a process. But typical examples of processes are programs that are meant to be running 'forever', like operating systems or controllers. Often they consist of several processes that run in parallel, with appropriate synchronisation between them. The proper way to describe such processes is not via input-output relations, like for sequential programs. Rather, one looks at their behaviour, represented as suitable (infinite) trees.

Let us start with the kind of example that is often used to introduced processes. Suppose we wish to describe a machine that can change €5 and €10 notes into €1 and €2 coins. We shall simply use '5' and '10' as input labels. And as output labels we use pairs (i, j) to describe the return of i 2-€ coins and j 1-€ coins. Also there is a special output action empty that indicates that the machine does not have enough coins left. Our abstract description will not determine which combination of coins is returned, but only gives the various options as a non-deterministic choice. Pictorially this yields a "state-transition" diagram like in Figure 3.5. Notice that the machine can only make a '5' or '10' transition if it can return a corresponding change. Otherwise, it can only do an 'empty' step.

In this section we shall describe such transition systems as coalgebras, namely as coalgebras of the functor $X \mapsto \mathcal{P}_{\mathrm{fin}}(X)^A$, for various sets A of "labels" or "actions". As usual, for states z, w and actions $a \in A$ we write $z \xrightarrow{a} w$ for $w \in c(z)(a)$, where $c\colon Y \to \mathcal{P}_{\mathrm{fin}}(Y)^A$ is our coalgebra. Note that for each z and a there are only finitely many successor states w with $z \xrightarrow{a} w$. Therefore, such transition systems are often called **finitely branching**.

In the above example, the set of labels is

$$E \;=\; \{5, 10, (2,1), (1,3), (0,5), (5,0), (4,2), (3,4), (2,6), (1,8), (0,10), \mathsf{empty}\}.$$

And as set of states we use

$$S \;=\; \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$$

with "change" coalgebra structure $\mathsf{ch}\colon S \to \mathcal{P}_{\mathrm{fin}}(S)^E$ given by:

$$\begin{array}{rcll}
\mathsf{ch}(s_0) & = & \lambda y \in E. \begin{cases} \{s_1, s_2, s_3\} & \text{if } y = 5 \\ \{s_4, s_5, s_6, s_7, s_8, s_9\} & \text{if } y = 10 \\ \{s_{10}\} & \text{if } y = \mathsf{empty} \\ \emptyset & \text{otherwise} \end{cases} \\
\mathsf{ch}(s_1) & = & \lambda y \in E. \begin{cases} \{s_0\} & \text{if } y = (2,1) \\ \emptyset & \text{otherwise} \end{cases} \\
\mathsf{ch}(s_2) & = & \lambda y \in E. \begin{cases} \{s_0\} & \text{if } y = (1,3) \\ \emptyset & \text{otherwise} \end{cases} \\
\multicolumn{4}{l}{\quad \textit{etc.}}
\end{array} \tag{3.2}$$

Since the functor $X \mapsto \mathcal{P}_{\mathrm{fin}}(X)^A$, for an arbitrary set A, is a finite polynomial functor, we know by Theorem 2.3.9 that it has a final coalgebra. In this section we shall write this final coalgebra as:

$$Z_A \xrightarrow[\cong]{\zeta_A} \mathcal{P}_{\mathrm{fin}}(Z_A)^A \tag{3.3}$$

We do not really care what these sets Z_A actually look like, because we shall only use their universal properties, and do not wish to depend on a concrete representation. However, concrete descriptions in terms of certain finitely branching trees modulo bisimilarity may be given, see [233, Section 4.3] (following [29]). In this context we shall call the elements of carrier Z_A of the final coalgebra **processes**.

Our change machine coalgebra ch with its set of actions E thus gives rise to a behaviour map:

$$\begin{array}{ccc}
\mathcal{P}_{\mathrm{fin}}(S)^E & \xrightarrow{\mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_{\mathsf{ch}})^E} & \mathcal{P}_{\mathrm{fin}}(Z_E)^E \\
{\scriptstyle \mathsf{ch}}\uparrow & & \cong\uparrow{\scriptstyle \zeta_E} \\
S & \xrightarrow[\mathsf{beh}_{\mathsf{ch}}]{} & Z_E
\end{array}$$

The behaviour function $\mathsf{beh}_{\mathsf{ch}}$ thus turns the concrete states $s_0, \ldots, s_{10}$ of our change machine into abstract states of the final coalgebra Z_E—*i.e.* into processes—with the same behaviour.

3.5.1 Process descriptions

Several languages have been proposed in the literature to capture processes, such as Algebra of Communicating Processes (ACP) [38, 75], Calculus of Communicating Systems (CCS) [179, 180] or Communicating Sequential Processes (CSP) [119]. The goal of such languages is to study processes via axiom systems in which notions like sequential and parallel execution, alternative choice, communication, *etc.* are formlised by means of algebraic operations and equations. It is not our aim to go into precise syntax and into the various differences between these formalisms. Instead we concentrate on the semantics and describe only a few central operations on processes, showing how they can be interpreted in a final coalgebra of the form (3.3). As an example, the above change machine could be described via a (recursive) process equation:

$$\begin{array}{rcl}
\mathsf{CH} & = & 5 \cdot (2,1) \cdot \mathsf{CH} + 5 \cdot (1,3) \cdot \mathsf{CH} + 5 \cdot (0,5) \cdot \mathsf{CH} + \\
& & 10 \cdot (5,0) \cdot \mathsf{CH} + 10 \cdot (4,2) \cdot \mathsf{CH} + 10 \cdot (3,4) \cdot \mathsf{CH} + \\
& & 10 \cdot (2,6) \cdot \mathsf{CH} + 10 \cdot (1,8) \cdot \mathsf{CH} + 10 \cdot (0,10) \cdot \mathsf{CH} + \\
& & \mathsf{empty} \cdot \mathsf{0}.
\end{array} \tag{3.4}$$

Process algebras can be understood more generally as providing a convenient syntax for describing various kinds of transition systems, see also Example 3.5.1 below.

In the following we fix an arbitrary set A of actions, and we consider the associated final coalgebra Z_A. We shall describe a collection of operations (such as $+$ and $\cdot$ above) on processes, understood as elements of Z_A.

The null process

One can define a trivial process which does nothing. It is commonly denoted as $\mathsf{0}$, and is defined as:

$$\mathsf{0} \;\stackrel{\text{def}}{=}\; \zeta_A^{-1}(\lambda y \in A.\, \emptyset).$$

This means that there are no successor states of the process $\mathsf{0} \in Z_A$—since by construction the set of successors $\zeta_A(\mathsf{0})(a)$ is empty, for each label $a \in A$.

Sum of two processes

Given two processes $z_1, z_2 \in Z_A$, one can define a sum process $z_1 + z_2 \in Z_A$ via the union operation $\cup$ on subsets:

$$z_1 + z_2 \;\stackrel{\text{def}}{=}\; \zeta_A^{-1}\big(\lambda y \in A.\, \zeta_A(z_1)(y) \cup \zeta_A(z_2)(y)\big).$$

This means that for each $w \in Z_A$ and $a \in A$, there is a transition $(z_1 + z_2) \xrightarrow{a} w$ out of a sum process if and only if there is either a transition $z_1 \xrightarrow{a} w$ or a transition $z_2 \xrightarrow{a} w$. In the process literature one usually encounters the following two rules:

$$\frac{z_1 \xrightarrow{a} w}{z_1 + z_2 \xrightarrow{a} w} \qquad \frac{z_2 \xrightarrow{a} w}{z_1 + z_2 \xrightarrow{a} w} \tag{3.5}$$

It is not hard to see that the structure $(Z_A, +, 0)$ is a commutative monoid with idempotent operation: $z + z = z$ for all $z \in Z_A$.

Prefixing of actions

Given a process $z \in Z_A$ and an action $a \in A$ there is a process $a \cdot z$ which first performs a and then continues with z. It can be defined as:

$$a \cdot z \;\stackrel{\text{def}}{=}\; \zeta_A^{-1}\big(\lambda y \in A.\, \mathsf{if}\ y = a\ \mathsf{then}\ \{z\}\ \mathsf{else}\ \emptyset\big).$$

We then have, for $w \in Z_A$ and $b \in A$,

$$\begin{array}{rcl}
(a \cdot z) \xrightarrow{b} w & \iff & w \in \zeta_A(a \cdot z)(b) \\
& \iff & w \in \big(\mathsf{if}\ b = a\ \mathsf{then}\ \{z\}\ \mathsf{else}\ \emptyset\big) \\
& \iff & b = a \wedge w = z.
\end{array}$$

This gives the standard rule:

$$\frac{}{a \cdot z \xrightarrow{a} z} \tag{3.6}$$

3.5.1. Example (Change machine, continued). Having defined prefixing and sum we can verify the validity of the change machine equation (3.4), for the interpretation $\mathsf{CH} = \mathsf{beh}_{\mathsf{ch}}(s_0) \in Z_E$ from (3.2). First we note that:

$$\begin{array}{rcl}
(2,1)\cdot\mathsf{CH} & = & \zeta_E^{-1}\Big(\lambda y \in E.\,\big(\mathsf{if}\ y = (2,1)\ \mathsf{then}\ \{\mathsf{beh}_{\mathsf{ch}}(s_0)\}\ \mathsf{else}\ \emptyset\big)\Big) \\
& = & \zeta_E^{-1}\Big(\lambda y \in E.\,\mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_{\mathsf{ch}})\big(\mathsf{if}\ y = (2,1)\ \mathsf{then}\ \{s_0\}\ \mathsf{else}\ \emptyset\big)\Big) \\
& = & \zeta_E^{-1}\Big(\lambda y \in E.\,\mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_{\mathsf{ch}})\big(\mathsf{ch}(s_1)(y)\big)\Big) \\
& = & \zeta_E^{-1}\Big(\lambda y \in E.\,\zeta_E\big(\mathsf{beh}_{\mathsf{ch}}(s_1)\big)(y)\Big) \\
& = & \mathsf{beh}_{\mathsf{ch}}(s_1)
\end{array}$$

Similar equations can be derived for the states $s_2, \ldots, s_9$. And for s_{10} we have:

$$\begin{array}{rcl}
\mathsf{beh}_{\mathsf{ch}}(s_{10}) & = & \zeta_E^{-1}\Big(\lambda y \in E.\,\zeta_E\big(\mathsf{beh}_{\mathsf{ch}}(s_{10})\big)(y)\Big) \\
& = & \zeta_E^{-1}\Big(\lambda y \in E.\,\mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_{\mathsf{ch}})\big(\mathsf{ch}(s_{10})(y)\big)\Big) \\
& = & \zeta_E^{-1}\Big(\lambda y \in E.\,\mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_{\mathsf{ch}})(\emptyset)\Big) \\
& = & \zeta_E^{-1}\Big(\lambda y \in E.\,\emptyset\Big) \\
& = & \mathsf{0}.
\end{array}$$

Finally we can check that the equation (3.4) holds for the behaviour $\mathsf{CH} = \mathsf{beh}_{\mathsf{ch}}(s_0)$.

$$\begin{array}{rcl}
\zeta_E(\mathsf{CH})(y) & & \\
& = & \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_{\mathsf{ch}})\big(\mathsf{ch}(s_0)(y)\big) \\
& = & \begin{cases}
\{\mathsf{beh}_{\mathsf{ch}}(s_1), \mathsf{beh}_{\mathsf{ch}}(s_2), \mathsf{beh}_{\mathsf{ch}}(s_3)\} & \text{if } y = 5 \\
\{\mathsf{beh}_{\mathsf{ch}}(s_4), \mathsf{beh}_{\mathsf{ch}}(s_5), \mathsf{beh}_{\mathsf{ch}}(s_6), & \\
\quad \mathsf{beh}_{\mathsf{ch}}(s_7), \mathsf{beh}_{\mathsf{ch}}(s_8), \mathsf{beh}_{\mathsf{ch}}(s_9)\} & \text{if } y = 10 \\
\{\mathsf{beh}_{\mathsf{ch}}(s_{10})\} & \text{if } y = \mathsf{empty}
\end{cases} \\
& = & \big(\mathsf{if}\ y = 5\ \mathsf{then}\ \{(2,1)\cdot\mathsf{CH}, (1,3)\cdot\mathsf{CH}, (0,5)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = 10\ \mathsf{then}\ \left\{\begin{array}{l}(5,0)\cdot\mathsf{CH}, (4,2)\cdot\mathsf{CH}, (3,4)\cdot\mathsf{CH}, \\ (2,6)\cdot\mathsf{CH}, (1,8)\cdot\mathsf{CH}, (0,10)\cdot\mathsf{CH}\end{array}\right\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = \mathsf{empty}\ \mathsf{then}\ \{\mathsf{0}\}\ \mathsf{else}\ \emptyset\big). \\
& = & \big(\mathsf{if}\ y = 5\ \mathsf{then}\ \{(2,1)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = 5\ \mathsf{then}\ \{(1,3)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = 5\ \mathsf{then}\ \{(0,5)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = 10\ \mathsf{then}\ \{(5,0)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = 10\ \mathsf{then}\ \{(4,2)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = 10\ \mathsf{then}\ \{(3,4)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = 10\ \mathsf{then}\ \{(2,6)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = 10\ \mathsf{then}\ \{(1,8)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = 10\ \mathsf{then}\ \{(0,10)\cdot\mathsf{CH}\}\ \mathsf{else}\ \emptyset\big) \cup \\
& & \big(\mathsf{if}\ y = \mathsf{empty}\ \mathsf{then}\ \{\mathsf{0}\}\ \mathsf{else}\ \emptyset\big) \\
& = & \zeta_E(5\cdot(2,1)\cdot\mathsf{CH})(y) \cup \zeta_E(5\cdot(1,3)\cdot\mathsf{CH})(y) \cup \\
& & \zeta_E(5\cdot(0,5)\cdot\mathsf{CH})(y) \cup \zeta_E(10\cdot(5,0)\cdot\mathsf{CH})(y) \cup \\
& & \zeta_E(10\cdot(4,2)\cdot\mathsf{CH})(y) \cup \zeta_E(10\cdot(3,4)\cdot\mathsf{CH})(y) \cup \\
& & \zeta_E(10\cdot(2,6)\cdot\mathsf{CH})(y) \cup \zeta_E(10\cdot(1,8)\cdot\mathsf{CH})(y) \cup \\
& & \zeta_E(10\cdot(0,10)\cdot\mathsf{CH})(y) \cup \zeta_E(\mathsf{empty}\cdot\mathsf{0})(y) \\
& = & \zeta_E\Big(5\cdot(2,1)\cdot\mathsf{CH} + 5\cdot(1,3)\cdot\mathsf{CH} + 5\cdot(0,5)\cdot\mathsf{CH} + \\
& & \quad 10\cdot(5,0)\cdot\mathsf{CH} + 10\cdot(4,2)\cdot\mathsf{CH} + 10\cdot(3,4)\cdot\mathsf{CH} + \\
& & \quad 10\cdot(2,6)\cdot\mathsf{CH} + 10\cdot(1,8)\cdot\mathsf{CH} + 10\cdot(0,10)\cdot\mathsf{CH} + \mathsf{empty}\cdot\mathsf{0}\Big)(y).
\end{array}$$

This example illustrates an approach for verifying that a transition system on an arbitrary state space satisfies a certain process equation:

1. Map the transition system to the final coalgebra via the behaviour map beh;
2. Check the equation for $\mathsf{beh}(s_0)$, where s_0 is a suitable (initial) state, using the above interpretations of the process combinators $+$, $\mathsf{0}$, $\cdot$ *etc.* on the final coalgebra.

3.5.2 A simple process algebra

In the previous subsection we have seen several process combinators, described as functions on a terminal coalgebra $Z_A \xrightarrow{\cong} \mathcal{P}_{\mathrm{fin}}(Z_A)^A$. Next we shall consider these basic combinators as constructors of a very simple process language, often called Basic Process Algebra (BPA), see [75, 231]. In the spirit of this text, the language of finite terms will be described as an initial algebra.

For an arbitrary set A of actions, consider the simple polynomial functor $\Sigma_A \colon \mathbf{Sets} \to \mathbf{Sets}$ given by

$$\Sigma_A(X) \;=\; 1 + (A \times X) + (X \times X)$$

An algebra $\Sigma_A(X) \to X$ for this functor thus consists of three operations which we call $0 \colon 1 \to X$ for null-process, $\cdot \colon A \times X \to X$ for prefixing, and $+ \colon X \times X \to X$ for sum. We have seen that the final coalgebra $Z_A \xrightarrow{\cong} \mathcal{P}_{\mathrm{fin}}(Z_A)^A$ carries a Σ_A-algebra structure which we shall write as $\xi_A \colon \Sigma_A(Z_A) \to Z_A$. It is given by the structure described earlier (before Example 3.5.1). Thus we a bialgebra of processes:

$$\Sigma_A(Z_A) \xrightarrow{\xi_A} Z_A \xrightarrow[\cong]{\zeta_A} \mathcal{P}_{\mathrm{fin}}(Z_A)^A$$

The free Σ_A-algebra on a set V consists the terms build up from the elements from V as variables. An interesting algebra is given by the free Σ_A-algebra on the final coalgebra Z_A. It consists of terms built out of processes, as studied in [212] under the phrase "processes as terms". Here we shall write P_A for the initial Σ_A-algebra, *i.e.* the free algebra on the empty set. The set P_A contains the "closed" process terms, without free variables. They are built up from 0 and $a \in A$. We shall write this algebra as $\alpha \colon \Sigma_A(P_A) \xrightarrow{\cong} P_A$. It is not hard to see that the set P_A of process terms also carries a bialgebra structure:

$$\Sigma_A(P_A) \xrightarrow[\cong]{\alpha} P_A \xrightarrow{\beta} \mathcal{P}_{\mathrm{fin}}(P_A)^A$$

The coalgebra β is defined by induction, following the transition rules (3.5), (3.6). For each $a \in A$,

$$\begin{array}{rcl} \beta(0)(a) &=& \emptyset \\ \beta(b \cdot s)(a) &=& \{s \mid b = a\} \\ \beta(s+t)(a) &=& \beta(s)(a) \cup \beta(t)(a). \end{array}$$

These definitions form a very simple example of a structural operations semantics (SOS): operational behaviour defined by induction on the structure of terms.

The next result shows that the denotational semantics given by initiality and operational semantics given by finality for process terms coincide.

3.5.2. Proposition. *In the above situation we obtain two maps $P_A \to Z_A$, one by initiality and one by finality:*

$$\begin{array}{ccc} \Sigma_A(P_A) & \xrightarrow{\Sigma_A(\mathsf{int}_\alpha)} & \Sigma_A(Z_A) \\ {\scriptstyle\alpha}\downarrow{\scriptstyle\cong} & & \downarrow{\scriptstyle\xi} \\ P_A & \xrightarrow[\mathsf{int}_\alpha]{} & Z_A \end{array} \qquad\qquad \begin{array}{ccc} \mathcal{P}_{\mathrm{fin}}(P_A)^A & \xrightarrow{\mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)^A} & \mathcal{P}_{\mathrm{fin}}(Z_A)^A \\ {\scriptstyle\beta}\uparrow & & \uparrow{\scriptstyle\cong}{\scriptstyle\zeta} \\ P_A & \xrightarrow[\mathsf{beh}_\beta]{} & Z_A \end{array}$$

These two maps are equal, so that $\mathsf{int}_\alpha = \mathsf{beh}_\beta \colon P_A \to Z_A$ is a "map of bialgebras" commuting both with the algebra and coalgebra structures. This proves in particular that the behavioural semantics beh_β of processes is compositional: it commutes with the term forming operations.

Proof. By induction on the structure of a term $s \in P_A$ we prove that $\mathsf{beh}_\beta(s) = \mathsf{int}_\alpha(s)$,

or equivalently, $\zeta(\mathsf{beh}_\beta(s))(a) = \zeta(\mathsf{int}_\alpha(s))(a)$, for all $a \in A$.

$$\begin{array}{rcll} \zeta(\mathsf{beh}_\beta(0))(a) &=& \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)^A\big(\beta(0)\big)(a) & \\ &=& \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)(\beta(0)(a)) & \\ &=& \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)(\emptyset) & \\ &=& \emptyset & \\ &=& \zeta(0)(a) & \\ &=& \zeta(\mathsf{int}_\alpha(0))(a). & \\ \zeta(\mathsf{beh}_\beta(b \cdot s))(a) &=& \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)(\beta(b \cdot s)(a)) & \\ &=& \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)(\{s \mid b = a\}) & \\ &=& \{\mathsf{beh}_\beta(s) \mid b = a\}) & \\ &\overset{\text{(IH)}}{=}& \{\mathsf{int}_\alpha(s) \mid b = a\}) & \\ &=& \zeta(b \cdot \mathsf{int}_\alpha(s))(a) & \\ &=& \zeta(\mathsf{int}_\alpha(b \cdot s))(a) & \\ \zeta(\mathsf{beh}_\beta(s+t))(a) &=& \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)(\beta(s+t)(a)) & \\ &=& \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)(\beta(s)(a) \cup \beta(t)(a)) & \\ &=& \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)(\beta(s)(a)) \cup \mathcal{P}_{\mathrm{fin}}(\mathsf{beh}_\beta)(\beta(t)(a)) & \\ &=& \zeta(\mathsf{beh}_\beta(s))(a) \cup \zeta(\mathsf{beh}_\beta(t))(a) & \\ &\overset{\text{(IH)}}{=}& \zeta(\mathsf{int}_\alpha(s))(a) \cup \zeta(\mathsf{int}_\alpha(t))(a) & \\ &=& \zeta(\mathsf{int}_\alpha(s) + \mathsf{int}_\alpha(t))(a) & \\ &=& \zeta(\mathsf{int}_\alpha(s+t))(a). & \end{array}$$

❑

3.5.3. Proposition. *Still in the above situation, the bisimilarity relation $\leftrightarrows$ on the set P_A of process terms is a congruence.*

Proof. Consider the following diagram, where we abbreviate $f = \mathsf{beh}_\beta = \mathsf{int}_\alpha$.

$$\begin{array}{ccccc} \Sigma_A(\leftrightarrows) & \xrightarrow{d} & \Sigma_A(P_A) \times \Sigma_A(P_A) & \overset{\Sigma_A(f \circ \pi_1)}{\underset{\Sigma_A(f \circ \pi_2)}{\rightrightarrows}} & \Sigma_A(Z_A) \\ \vdots & & {\scriptstyle\alpha \times \alpha}\downarrow & & \downarrow{\scriptstyle\xi} \\ \leftrightarrows & \xrightarrow[e]{} & P_A \times P_A & \overset{f \circ \pi_1}{\underset{f \circ \pi_2}{\rightrightarrows}} & Z_A \end{array}$$

The map e is the equaliser of $f \circ \pi_1$ and $f \circ \pi_2$, using Theorem 3.4.1. The map d is the pair $\langle \Sigma_A(e \circ \pi_1), \Sigma(e \circ \pi_2) \rangle$. We show that $(\alpha \times \alpha) \circ d$ equalises $f \circ \pi_1, f \circ \pi_2$. The dashed arrow then exists by the universal property of the equaliser e, making $\leftrightarrows$ a congruence (see Exercise 3.3.2). Thus, what remains is:

$$\begin{array}{rcl} f \circ \pi_1 \circ (\alpha \times \alpha) \circ d &=& \mathsf{int}_\alpha \circ \alpha \circ \pi_1 \circ d \\ &=& \xi \circ \Sigma_A(\mathsf{int}_\alpha) \circ \Sigma_A(e \circ \pi_1) \\ &=& \xi \circ \Sigma_A(\mathsf{int}_\alpha) \circ \Sigma_A(e \circ \pi_2) \\ &=& \mathsf{int}_\alpha \circ \alpha \circ \pi_2 \circ d \\ &=& f \circ \pi_2 \circ (\alpha \times \alpha) \circ d. \end{array}$$

❑

This result shows that $s \leftrightarrows s'$ and $t \leftrightarrows t'$ implies $a \cdot s \leftrightarrows a \cdot s'$ and $s + t \leftrightarrows s' + t'$. Such congruence results are fundamental in process algebra, because they show that the algebraic operations for process formation preserve indistinguishability of behaviour. Later, in Chapter 6, this topic will be studied in greater generality.

Exercises

3.5.1. Complete the definition of the coalgebra $\mathsf{ch}\colon S \to \mathcal{P}_{\mathrm{fin}}(S)^E$ in the beginning of this section.

3.5.2. Prove that $(Z_A, +, 0)$ is indeed a commutative monoid, as claimed above.

3.5.3. One can consider each action $a \in A$ as a process

$$\widehat{a} \stackrel{\text{def}}{=} a \cdot 0 \in Z_A$$

It can do only an a-transition. Prove that this yields an injection $A \hookrightarrow Z_A$.

3.5.4. Consider the following alternative process equation for a Euro change machine.

$$\begin{aligned} \mathsf{CH'} \;=\; & 5 \cdot \Big((2,1) \cdot \mathsf{CH'} + (1,3) \cdot \mathsf{CH'} + (0,5) \cdot \mathsf{CH'} \Big) \\ & + 10 \cdot \Big((5,0) \cdot \mathsf{CH'} + (4,2) \cdot \mathsf{CH'} + (3,4) \cdot \mathsf{CH'} + \\ & \qquad (2,6) \cdot \mathsf{CH'} + (1,8) \cdot \mathsf{CH'} + (0,10) \cdot \mathsf{CH'} \Big) \\ & + \mathsf{empty}. \end{aligned}$$

Understand the difference between CH in (3.4) and this CH', for instance by describing a suitable transition system which forms a model of this equation.
Are CH and CH' bisimilar?

Chapter 4

Invariants

The second important notion in the logic of coalgebras, beside bisimulation, is invariance. Whereas a bisimulation is a binary relation on state spaces that is closed under transitions, an invariant is a predicate, or unary relation if you like, on a state space which is closed. This means, once an invariant holds, it will continue to hold no matter which operations are applied. That is: coalgebras maintain their invariants.

Invariants are important in the description of systems, because they often express certain implicit assumptions, like: this integer value will always be non-zero (so that dividing by this integer is safe), or: the contents of this tank will never be below a given minimum value. Thus, invariants are "safety properties", which express that something bad will never happen.

This chapter will introduce a general notion of invariant for a coalgebra of a polynomial functor, via predicate lifting. Predicate lifting is the unary analogue of relation lifting. Various properties of invariants are established, in particular their intimate relation to subcoalgebras. An important application of invariants lies in a generic temporal logic for coalgebras, involving henceforth $\Box$ and eventually $\Diamond$ operators on predicates, that will be introduced in Section 4.3. Further, invariants play a rôle in the construction of final coalgebras, a topic that was postponed earlier in Section 2.3. The final section of this chapter will use the details of such a construction in trace semantics for coalgebras.

4.1 Predicate lifting

This section will introduce the technique of predicate lifting. It will be used in the next section in the definition of invariance for (co)algebras. Here we will first establish various elementary, useful properties of predicate lifting. Special attention will be paid to the left adjoint to predicate lifting, called predicate lowering.

Predicate lifting for a polynomial functor F is an operation $\mathrm{Pred}(F)$ which sends a predicate $P \subseteq X$ on a set X to a "lifted" predicate $\mathrm{Pred}(F)(P) \subseteq F(X)$ on the result of applying the functor F to X. The idea is that P should hold on all occurrences of X inside $F(X)$, as suggested in:

$$\begin{array}{rcl} F(X) & = & \boxed{\cdots \quad X \quad \cdots \quad A \quad \cdots \quad X \quad \cdots} \\ \mathrm{Pred}(F)(P) & = & \qquad\quad \big| \qquad\qquad\quad \big| \\ & & \qquad\quad P \qquad\qquad\quad P \end{array}$$

The formal definition proceeds by induction on the structure of the functor F, just like for relation lifting in Definition 3.1.1.

4.1.1. Definition (Predicate lifting). Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor, and let X be an arbitrary set. The mapping $\mathrm{Pred}(F)$ which sends a predicate $P \subseteq X$ to a "lifted" predicate $\mathrm{Pred}(F)(P) \subseteq F(X)$ is defined by induction on the structure of F, in accordance with the points in Definition 2.2.1.

(1) If F is the identity functor, then

$$\mathrm{Pred}(F)(P) \;=\; P$$

(2) If F is a constant functor $Y \mapsto A$, then

$$\mathrm{Pred}(F)(P) \;=\; \top_A \;=\; (A \subseteq A).$$

(3) If F is a product $F_1 \times F_2$, then

$$\mathrm{Pred}(F)(P) \;=\; \{(u,v) \mid \mathrm{Pred}(F_1)(P)(u) \wedge \mathrm{Pred}(F_2)(P)(v)\}$$

(4) If F is a coproduct $F_1 + F_2$, then

$$\mathrm{Pred}(F)(P) \;=\; \{\kappa_1(u) \mid \mathrm{Pred}(F_1)(P)(u)\} \cup \{\kappa_2(v) \mid \mathrm{Pred}(F_2)(P)(v)\}$$

(5) If F is an exponent G^A, then

$$\mathrm{Pred}(F)(P) \;=\; \{f \mid \forall a \in A.\, \mathrm{Pred}(G)(P)(f(a))\}$$

(6) If F is a powerset $\mathcal{P}(G)$, then

$$\mathrm{Pred}(F)(P) \;=\; \{U \mid \forall u \in U.\, \mathrm{Pred}(G)(P)(u)\}$$

This same formula will be used in case F is a *finite* powerset $\mathcal{P}_{\mathrm{fin}}(G)$.

(7) If F is a list $G^\star$, then

$$\mathrm{Pred}(F)(P) \;=\; \{\langle u_1, \ldots, u_n\rangle \mid \forall i \leq n.\, \mathrm{Pred}(G)(P)(u_i)\}.$$

First we show that predicate and relation lifting are closely related.

4.1.2. Lemma. (i) *Relation lifting $\mathrm{Rel}(F)$ and predicate lifting $\mathrm{Pred}(F)$ for a polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$ are related in the following way.*

$$\mathrm{Rel}(F)(\textstyle\coprod_{\delta_X}(P)) \;=\; \textstyle\coprod_{\delta_{F(X)}}(\mathrm{Pred}(F)(P)),$$

where $\delta_X = \langle \mathrm{id}_X, \mathrm{id}_X\rangle$ and so $\coprod_{\delta_X}(P) = \{\delta_X(x,x) \mid x \in P\} = \{(x,x) \mid x \in P\}$.

(ii) *Similarly,*

$$\mathrm{Pred}(F)(\textstyle\coprod_{\pi_i}(R)) \;=\; \textstyle\coprod_{\pi_i}(\mathrm{Rel}(F)(R)),$$

where $\coprod_{\pi_1}(R) = \{x_1 \mid \exists x_2.\, R(x_1,x_2)\}$ is the domain of the relation R, and $\coprod_{\pi_2}(R) = \{x_2 \mid \exists x_1.\, R(x_1,x_2)\}$ is its codomain.

(iii) *As a result, predicate lifting can be expressed in terms of relation lifting:*

$$\mathrm{Pred}(F)(P) \;=\; \textstyle\coprod_{\pi_i}\big(\mathrm{Rel}(F)(\coprod_\delta(P))\big)$$

for both $i = 1$ and $i = 2$.

Proof. (i) + (ii) By induction on the structure of F.

(iii) Since $\coprod_{\pi_i} \circ \coprod_\delta = \mathrm{id}$ we use the previous point to get:

$$\mathrm{Pred}(F)(P) \;=\; \textstyle\coprod_{\pi_i}\coprod_\delta \mathrm{Pred}(F)(P) \;=\; \coprod_{\pi_i}\mathrm{Rel}(F)(\coprod_\delta P).$$ ❑

Despite this last result, it is useful to study predicate lifting separately, because it has some special properties that relation lifting does not enjoy—like preservation of intersections, see the next result, and thus existence of a left adjoint, see Subsection 4.1.1.

4.1.3. Lemma. *Predicate lifting $\mathrm{Pred}(F)$ w.r.t. a polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$ satisfies the following properties.*

(i) *It preserves arbitrary intersections: for every collection of predicates $(P_i \subseteq X)_{i\in I}$,*

$$\mathrm{Pred}(F)(\textstyle\bigcap_{i\in I} P_i) \;=\; \bigcap_{i\in I} \mathrm{Pred}(F)(P_i).$$

A special case (intersection over $I = \emptyset$) worth mentioning is preservation of truth:

$$\mathrm{Pred}(F)(\top_X) \;=\; \top_{FX}.$$

Another consequence is that predicate lifting is monotone:

$$P \subseteq Q \;\Longrightarrow\; \mathrm{Pred}(F)(P) \subseteq \mathrm{Pred}(F)(Q).$$

(ii) *It preserves inverse images: for a function $f\colon X \to Y$ and predicate $Q \subseteq Y$,*

$$\mathrm{Pred}(F)(f^{-1}(Q)) \;=\; F(f)^{-1}(\mathrm{Pred}(F)(Q)).$$

(iii) *Relation lifting also preserves direct images: for $f\colon X \to Y$ and $P \subseteq X$,*

$$\mathrm{Pred}(F)(\textstyle\coprod_f(P)) \;=\; \coprod_{F(f)}(\mathrm{Pred}(F)(P)).$$

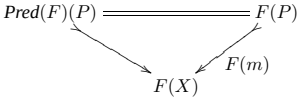
Proof. (i) + (ii) By induction on the structure of F.

(iii) This is easily seen via the link with relation lifting:

$$\begin{array}{lcll}
\mathrm{Pred}(F)(\coprod_f P) & = & \coprod_{\pi_1}\mathrm{Rel}(F)(\coprod_\delta \coprod_f P) & \text{by Lemma 4.1.2 (iii)} \\
& = & \coprod_{\pi_1}\mathrm{Rel}(F)(\coprod_{f\times f}\coprod_\delta P) & \\
& = & \coprod_{\pi_1}\coprod_{Ff\times Ff}\mathrm{Rel}(F)(\coprod_\delta P) & \text{by Lemma 3.2.2 (ii)} \\
& = & \coprod_{Ff}\coprod_{\pi_1}\mathrm{Rel}(F)(\coprod_\delta P) & \\
& = & \coprod_{F(f)}(\mathrm{Pred}(F)(P)), & \text{again by Lemma 4.1.2 (iii).}
\end{array}$$ ❑

The next result is the analogue for predicate lifting of Lemma 3.3.1 for relation lifting. It relies on considering predicates as sets themselves, and may be understood as: predicate lifting commutes with comprehension, see Exercise 4.2.4 (iii). This is described in a more general logical setting in [117]. But in the current set-theoretic setting the connection between predicate lifting and functor application is extremely simple.

4.1.4. Lemma. *For a polynomial functor F, predicate lifting $\mathrm{Pred}(F)(P)$ for a predicate $m\colon P \rightarrowtail X$ the same as functor application in:*



Note that from the fact that polynomial functors preserve weak pullbacks (Lemma 3.3.6) we can already conclude that $F(m)$ is a mono, see Exercise 3.3.7 (i).

Proof. Formally, one proves for $z \in F(X)$,

$$z \in \mathrm{Pred}(F)(P) \iff \exists! z' \in F(P).\, F(m)(z') = z.$$

This is obtained by induction on the structure of the polynomial functor F. ❑

In view of this result one may wish to define for an arbitrary functor $G\colon \mathbb{C} \to \mathbb{C}$ which preserves monomorphisms (such as a weak pullback preserving functor, see Exercise 3.3.7), an associated predicate lifting operation $\mathrm{Pred}(G)(-)$ simply via functor application:

$$\Big(P \rightarrowtail^{m} X\Big) \qquad \longmapsto \qquad \Big(G(P) \rightarrowtail^{G(m)} G(X)\Big) \tag{4.1}$$

More generally, if G does not preserve monos we may define $\mathrm{Pred}(G)$ via factorisation:

$$\begin{array}{ccc} P & & \mathrm{Pred}(G)(P) \\ {\scriptstyle m}\downarrow & \longmapsto & \downarrow \\ X & & G(X) \end{array} \qquad\qquad \begin{array}{ccc} \mathrm{Pred}(G)(P) & \twoheadleftarrow & G(P) \\ \downarrow & \swarrow_{G(m)} & \\ G(X) & & \end{array}$$

This operation automatically preserves truth predicates (given by identity monomorphism $X \rightarrowtail X$), see Lemma 4.1.3 (i). Further properties like preservation of intersections may be assumed as explicit requirements, when needed. This leads to a more axiomatic approach. A still further generalisation is possible by moving to fibrations, and consider predicate lifting as a suitably axiomatised functor between categories of predicates (like in Exercise 4.2.4 (iii)).

Here we shall stick to our inductive approach to predicate lifting for polynomial functors on $\mathbf{Sets}$ because it gives the best handle on concrete examples.

4.1.1 Predicate lowering as liftings left adjoint

We conclude this section with a Galois connection involving predicate lifting. In the next section we shall use predicate lifting for a next-time operator $\bigcirc$ in a temporal logic of coalgebras. There is also a lasttime operator $\underset{\leftarrow}{\bigcirc}$ (see Subsection 4.3.1), for which we shall need this left adjoint (or lower Galois adjoint) to predicate lifting $\mathrm{Pred}(F)$. We shall write this left (or lower) adjoint as $\underset{\leftarrow}{\mathrm{Pred}}(F)$, and shall call it "predicate lowering". Such an adjoint must exist because predicate lifting preserves arbitrary intersections, see Lemma 4.1.3 (i)—see *e.g.* [176] or [150, I, Theorem 4.2]. But it can also be defined explicitly by induction on the structure of the functor. This is what we shall do.

4.1.5. Proposition (From [129, 136]). *Predicate lifting for a polynomial functor F forms a monotone function $\mathit{Pred}(F)\colon \mathcal{P}(X) \to \mathcal{P}(F(X))$ between powerset posets. In the opposite direction there is also an operation $\underset{\leftarrow}{\mathit{Pred}}(F)\colon \mathcal{P}(F(X)) \to \mathcal{P}(X)$ satisfying*

$$\underset{\leftarrow}{\mathit{Pred}}(F)(Q) \subseteq P \iff Q \subseteq \mathit{Pred}(F)(P)$$

Hence $\underset{\leftarrow}{\mathit{Pred}}(F)$ is the left adjoint of $\mathit{Pred}(F)$ in a Galois connection $\underset{\leftarrow}{\mathit{Pred}}(F) \dashv \mathit{Pred}(F)$.

Proof. One can define $\underset{\leftarrow}{\mathrm{Pred}}(F)(Q) \subseteq X$ for $Q \subseteq F(X)$ by induction on the structure of F.

(1) If F is the identity functor, then

$$\underset{\leftarrow}{\mathrm{Pred}}(F)(Q) \;=\; Q.$$

(2) If F is a constant functor $Z \mapsto A$, then

$$\underset{\leftarrow}{\mathrm{Pred}}(F)(Q) \;=\; \bot_A \;=\; (\emptyset \subseteq A).$$

(3) If F is a product $F_1 \times F_2$, then

$$\begin{array}{rcl} \underset{\leftarrow}{\mathrm{Pred}}(F)(Q) & = & \underset{\leftarrow}{\mathrm{Pred}}(F_1)(\coprod_{\pi_1}(Q)) \cup \underset{\leftarrow}{\mathrm{Pred}}(F_2)(\coprod_{\pi_2}(Q)) \\ & = & \underset{\leftarrow}{\mathrm{Pred}}(F_1)(\{u \in F_1(X) \mid \exists v \in F_2(X).\, Q(u,v)\}) \\ & & \quad \cup\, \underset{\leftarrow}{\mathrm{Pred}}(F_2)(\{v \in F_2(X) \mid \exists u \in F_1(X).\, Q(u,v)\}). \end{array}$$

(4) If F is a coproduct $F_1 + F_2$, then

$$\begin{array}{rcl} \underset{\leftarrow}{\mathrm{Pred}}(F)(Q) & = & \underset{\leftarrow}{\mathrm{Pred}}(F_1)(\kappa_1^{-1}(Q)) \cup \underset{\leftarrow}{\mathrm{Pred}}(F_2)(\kappa_2^{-1}(Q)) \\ & = & \underset{\leftarrow}{\mathrm{Pred}}(F_1)(\{u \mid Q(\kappa_1(u))\}) \cup \underset{\leftarrow}{\mathrm{Pred}}(F_2)(\{v \mid Q(\kappa_2(v))\}). \end{array}$$

(5) If F is an exponent G^A, then

$$\underset{\leftarrow}{\mathrm{Pred}}(F)(Q) \;=\; \underset{\leftarrow}{\mathrm{Pred}}(G)(\{f(a) \mid a \in A \wedge Q(f)\})$$

(6) If F is a powerset $\mathcal{P}(G)$, then

$$\underset{\leftarrow}{\mathrm{Pred}}(F)(Q) \;=\; \underset{\leftarrow}{\mathrm{Pred}}(G)(\bigcup Q).$$

This same formula will be used in case F is a *finite* powerset $\mathcal{P}_{\mathrm{fin}}(G)$.

(7) If F is a list $G^\star$, then

$$\underset{\leftarrow}{\mathrm{Pred}}(F)(Q) \;=\; \underset{\leftarrow}{\mathrm{Pred}}(G)(\{u_i \mid Q(\langle u_1, \ldots, u_n\rangle)\}).$$ ❑

Being a left adjoint means that functions $\underset{\leftarrow}{\mathrm{Pred}}(F)$ preserve certain "colimit" structures.

4.1.6. Lemma. *Let F be a polynomial functor. Its operations $\underset{\leftarrow}{\mathit{Pred}}(F)\colon \mathcal{P}(F(X)) \to \mathcal{P}(X)$ preserve:*
(i) *unions $\bigcup$ of predicates;*
(ii) *direct images $\coprod$, in the sense that for $f\colon X \to Y$,*

$$\underset{\leftarrow}{\mathit{Pred}}(F)(\coprod_{F(f)}(Q)) \;=\; \coprod_f \underset{\leftarrow}{\mathit{Pred}}(F)(Q).$$

Proof. (i) This is a general property of left (Galois) adjoints, as illustrated in the beginning of Section 2.5.

(ii) One can use a "composition of adjoints" argument, or reason directly with the adjunctions:

$$\begin{array}{rcl} \underset{\leftarrow}{\mathrm{Pred}}(F)(\coprod_{F(f)}(Q)) \;\subseteq\; P & \iff & \coprod_{F(f)}(Q) \;\subseteq\; \mathrm{Pred}(F)(P) \\ & \iff & Q \;\subseteq\; F(f)^{-1}\mathrm{Pred}(F)(P) = \mathrm{Pred}(F)(f^{-1}(P)) \\ & & \qquad \text{by Lemma 4.1.3 (ii)} \\ & \iff & \underset{\leftarrow}{\mathrm{Pred}}(F)(Q) \;\subseteq\; f^{-1}(Q) \\ & \iff & \coprod_f \underset{\leftarrow}{\mathrm{Pred}}(F)(Q) \;\subseteq\; P. \end{array}$$

❑

This left adjoint to predicate lifting gives rise to a special kind of mapping from an arbitrary polynomial functor to the powerset functor. With this mapping each coalgebra can be turned into an unlabeled transition system.

4.1.7. Proposition. *For a polynomial functor F and a set X, write $\sigma_X \colon F(X) \to \mathcal{P}(X)$ for the following composite.*

$$\sigma_X \stackrel{\text{def}}{=} \Big(F(X) \xrightarrow{\{-\}} \mathcal{P}(F(X)) \xrightarrow{\underleftarrow{\text{Pred}}(F)} \mathcal{P}(X) \Big)$$

This collection $(\sigma_X)_X$ of maps indexed by sets X is "natural" in the sense that for each function $f \colon X \to Y$ the following diagram commutes.

$$\begin{array}{ccc} F(X) & \xrightarrow{\sigma_X} & \mathcal{P}(X) \\ {\scriptstyle F(f)}\downarrow & & \downarrow{\scriptstyle \mathcal{P}(f) = \coprod_f} \\ F(Y) & \xrightarrow[\sigma_Y]{} & \mathcal{P}(Y) \end{array}$$

Proof. We only need to prove naturality. For $u \in F(X)$:

$$\begin{array}{rcll} \coprod_f \big(\sigma_X(u)\big) & = & \coprod_f \underleftarrow{\text{Pred}}(F)(\{u\}) & \\ & = & \underleftarrow{\text{Pred}}(F)(\coprod_{F(f)} \{u\}) & \text{by Lemma 4.1.6 (ii)} \\ & = & \underleftarrow{\text{Pred}}(F)(\{F(f)(u)\}) & \\ & = & \sigma_Y\big(F(f)(u)\big). & \end{array}$$

❑

Natural transformations play a rôle in the following fundamental translation result for coalgebras.

4.1.8. Proposition. *Let $\mathbb{C}$ be an arbitrary category. A natural transformation $\alpha \colon F \Rightarrow G$ between two endofunctors $F, G \colon \mathbb{C} \to \mathbb{C}$ gives rise to a functor between categories of coalgebras:*

$$\begin{array}{ccc} \mathbf{CoAlg}(F) & \xrightarrow{\alpha \circ (-)} & \mathbf{CoAlg}(G) \\ & \searrow \quad \swarrow & \\ & \mathbb{C} & \end{array}$$

which commutes with the forgetful functors to $\mathbb{C}$. It is given by:

$$\Big(X \xrightarrow{c} F(X) \Big) \longmapsto \Big(X \xrightarrow{\alpha_X \circ c} G(X) \Big) \qquad \textit{and} \qquad f \longmapsto f.$$

❑

By combining the previous two propositions we obtain a functor

$$\mathbf{CoAlg}(F) \longrightarrow \mathbf{CoAlg}(\mathcal{P}) \tag{4.2}$$

from coalgebras of a polynomial functor to unlabeled transition systems. This translation will be further investigated in Section 4.3. As can be seen in Exercise 4.1.2, the translation removes much of the structure of the coalgebra. However, it makes the intuitive idea precise that states of a coalgebra can make transitions.

Exercises

4.1.1. Use Lemmas 4.1.4 and 3.3.1 to check that relation lifting can also be expressed via predicate lifting. For a relation $\langle r_1, r_2 \rangle \colon R \rightarrowtail X \times Y$,

$$\text{Rel}(F)(R) \quad = \quad \coprod_{(F(r_1), F(r_2))} \text{Pred}(F)(R).$$

4.1.2. Let $X \xrightarrow{\langle \delta, \varepsilon \rangle} X^A \times B$ be a deterministic automaton. Prove that the associated unlabeled transition system, according to (4.2), is described by:

$$x \longrightarrow x' \iff \exists a \in A.\, \delta(x)(a) = x'.$$

4.1.3. For a polynomial functor $F \colon \mathbf{Sets} \to \mathbf{Sets}$, consider the forgetful functor $U \colon \mathbf{CoAlg}(F) \to \mathbf{Sets}$. Check that predicate lifting $\text{Pred}(F)$ is a natural tranformation

$$2^U \xRightarrow{\text{Pred}(F)} 2^{FU}$$

where $2 = \{0, 1\}$ and predicates $P \subseteq X$ are identifi ed with their characteristic functions $X \to 2$. The functor $2^U \colon \mathbf{CoAlg}(F) \to \mathbf{Sets}$ thus sends a coalgebra $X \to F(X)$ to the powerset $\mathcal{P}(X)$.
Predicate liftings are used in such natural transformation formulation in [190] as starting point for temporal logics, like in Section 4.3.

4.2 Invariants

This section will introduce invariants via predicate lifting (introduced in the previous section). It will first describe certain standard examples and results—especially relating invariants to subcoalgebras. Many of the results we describe for invariants occur in [216, Section 6], but with subcoalgebra terminology, and thus with slightly different proofs. Special attentions will be paid below to "greatest invariants" $\Box P$, for an arbitrary predicate P on the state spaces of a coalgebra. These greatest invariants are useful in various constructions. Most importantly in this section, in the construction of products for coalgebras. In the next section, the relevance of these greatest invariants will be discussed in the context of a temporal logic for coalgebras.

We shall define the notion of invariant, both for coalgebras and for algebras. In both cases it is a predicate which is closed under the operation. There does not seem to be an established (separate) terminology in algebra, so we use the same as in coalgebra.

4.2.1. Definition. Let $F \colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor.

(i) An **invariant** for a coalgebra $c \colon X \to F(X)$ is a predicate $P \subseteq X$ satisfying for all $x \in X$,

$$x \in P \implies c(x) \in \text{Pred}(F)(P)$$

Equivalently,

$$P \subseteq c^{-1}(\text{Pred}(F)(P)) \qquad \text{or} \qquad \coprod_c(P) \subseteq \text{Pred}(F)(P).$$

(ii) An **invariant** for an algebra $a \colon F(X) \to X$ is a predicate $P \subseteq X$ satisfying for all $u \in F(X)$,

$$u \in \text{Pred}(F)(P) \implies a(u) \in P$$

That is,

$$\text{Pred}(F)(P) \subseteq a^{-1}(P) \qquad \text{or} \qquad \coprod_a(\text{Pred}(F)(P)) \subseteq P.$$

This section concentrates on invariants for coalgebras, but occasionally invariants for algebras are also considered. We first relate invariants to bisimulations. There are similar results for congruences, see Exercise 4.2.1.

4.2.2. Lemma. *Consider two coalgebras $c\colon X \to F(X)$ and $d\colon Y \to F(Y)$ of a polynomial functor F. Then:*

(i) *if $R \subseteq X \times Y$ is a bisimulation, then both its domain $\coprod_{\pi_1} R = \{x \mid \exists y.\, R(x,y)\}$ and codomain $\coprod_{\pi_2} R = \{y \mid \exists x.\, R(x,y)\}$ are invariants.*

(ii) *an invariant $P \subseteq X$ yields a bisimulation $\coprod_\delta P = \{(x,x) \mid x \in P\} \subseteq X \times X$.*

Proof. (i) If the relation R is a bisimulation, then the predicate $\coprod_{\pi_1} R \subseteq X$ is an invariant, since:

$$\begin{array}{rcll} \coprod_c \coprod_{\pi_1} R & = & \coprod_{\pi_1} \coprod_{c\times d} R & \\ & \subseteq & \coprod_{\pi_1} \mathrm{Rel}(F)(R) & \text{because } R \text{ is a bisimulation} \\ & = & \mathrm{Pred}(F)(\coprod_{\pi_1} R) & \text{by Lemma 4.1.2 (ii).} \end{array}$$

Similarly, $\coprod_{\pi_2} R \subseteq Y$ is an invariant for the coalgebra d.

(ii) Suppose now that $P \subseteq X$ is an invariant. Then:

$$\begin{array}{rcll} \coprod_{c\times c} \coprod_\delta P & = & \coprod_\delta \coprod_c P & \\ & \subseteq & \coprod_\delta \mathrm{Pred}(F)(P) & \text{since } P \text{ is an invariant} \\ & = & \mathrm{Rel}(F)(\coprod_\delta P) & \text{by Lemma 4.1.2 (i).} \end{array}$$

❑

4.2.3. Example. We consider invariants for both deterministic and non-deterministic automata.

(i) As is well-known by now, a deterministic automaton $\langle \delta, \epsilon\rangle\colon X \to X^A \times B$ is a coalgebra for the functor $F = \mathrm{id}^A \times B$. Predicate lifting for this functor yields for a predicate $P \subseteq X$ a new predicate $\mathrm{Pred}(F)(P) \subseteq \big(X^A \times B\big)$, given by:

$$\mathrm{Pred}(F)(P)(f,b) \iff \forall a \in A.\, P(f(a)).$$

A predicate $P \subseteq X$ is thus an invariant w.r.t. the coalgebra $\langle \delta, \epsilon\rangle\colon X \to X^A \times B$ if, for all $x \in X$,

$$\begin{array}{rcl} P(x) & \Longrightarrow & \mathrm{Pred}(F)(P)\big((\delta(x), \epsilon(x))\big) \\ & \iff & \forall a \in A.\, P(\delta(x)(a)) \\ & \iff & \forall a \in A.\, \forall x' \in X.\, x \xrightarrow{a} x' \Rightarrow P(x'). \end{array}$$

Thus, once a state is in an invariant P, all its successor states are also in P. Once an invariant holds, it will continue to hold.

(ii) A non-deterministic automaton $\langle \delta, \epsilon\rangle\colon X \to \mathcal{P}(X)^A \times B$ is a coalgebra for the functor $F = \mathcal{P}(\mathrm{id})^A \times B$. Predicate lifting for this functor sends a predicate $P \subseteq X$ to the predicate $\mathrm{Pred}(F)(P) \subseteq \big(\mathcal{P}(X)^A \times B\big)$ given by:

$$\mathrm{Pred}(F)(P)(f,b) \iff \forall a \in A.\, \forall x' \in f(a).\, P(x')$$

This $P \subseteq X$ is then an invariant for the non-deterministic automaton $\langle \delta, \epsilon\rangle\colon X \to \mathcal{P}(X)^A \times B$ if for all $x \in X$,

$$\begin{array}{rcl} P(x) & \Longrightarrow & \mathrm{Pred}(F)(P)\big((\delta(x), \epsilon(x))\big) \\ & \iff & \forall a \in A.\, \forall x' \in \delta(x)(a).\, P(x') \\ & \iff & \forall a \in A.\, \forall x' \in X.\, x \xrightarrow{a} x' \Rightarrow P(x'). \end{array}$$

4.2.4. Proposition. *Let $X \xrightarrow{c} F(X)$ and $Y \xrightarrow{d} F(Y)$ be two coalgebras of a polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$.*

(i) *Invariants are closed under arbitrary unions and intersections: if predicates $P_i \subseteq X$ are invariants for $i \in I$, then their union $\bigcup_{i\in I} P_i$ and intersection $\bigcap_{i\in I} P_i$ are invariants.*

In particular, falsity $\bot$ (union over $I = \emptyset$) and truth $\top$ (intersection over $I = \emptyset$) are invariants.

(ii) *Invariants are closed under direct and inverse images along homomorphisms: if $f\colon X \to Y$ is a homomorphism of coalgebras, and $P \subseteq X$ and $Q \subseteq Y$ are invariants, then so are $\coprod_f(P) \subseteq Y$ and $f^{-1}(Q) \subseteq X$.*

In particular, the image $\mathrm{Im}(f) = \coprod_f(\top)$ of a homomorphism is an invariant.

Proof. (i) First we note that inverse images preserve both unions and intersections. Closure of invariants under unions then follows from monotonicty of predicate lifting: $P_i \subseteq c^{-1}(\mathrm{Pred}(F)(P_i)) \subseteq c^{-1}(\mathrm{Pred}(F)(\bigcup_{i\in I} P_i)$ for each $i \in I$, so that we may conclude $\bigcup_{i\in I} P_i \subseteq c^{-1}\mathrm{Pred}(F)(\bigcup_{i\in I} P_i)$. Similarly, closure under intersection follows because predicate lifting preserves intersections, see Lemma 4.1.3 (i).

(ii) For preservation of direct images assume that $P \subseteq X$ is an invariant. Then:

$$\begin{array}{rcll} \coprod_d \coprod_f P & = & \coprod_{F(f)} \coprod_c P & \text{because } f \text{ is a homomorphism} \\ & \subseteq & \coprod_{F(f)} \mathrm{Pred}(F)(P) & \text{since } P \text{ is an invariant} \\ & = & \mathrm{Pred}(F)(\coprod_f P) & \text{by Lemma 4.1.3 (iii).} \end{array}$$

Similarly, if $Q \subseteq Y$ is an invariant, then:

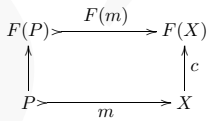
$$\begin{array}{rcll} f^{-1}(Q) & \subseteq & f^{-1}d^{-1}(\mathrm{Pred}(F)(Q)) & \text{because } Q \text{ is an invariant} \\ & = & c^{-1}F(f)^{-1}(\mathrm{Pred}(F)(Q)) & \text{because } f \text{ is a homomorphism} \\ & = & c^{-1}(\mathrm{Pred}(F)(f^{-1}(Q)) & \text{by Lemma 4.1.3 (ii).} \end{array}$$

❑

The next result readily follows from Lemma 4.1.4. It is the analogue of Theorem 3.3.2, and has important consequences.

4.2.5. Theorem. *Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor.*

(i) *A predicate $m\colon P \rightarrowtail X$ on the state space of a coalgebra $c\colon X \to F(X)$ is an invariant if and only if $P \rightarrowtail X$ is a **subcoalgebra**: there is a (necessarily unique) coalgebra structure $P \to F(P)$ making $m\colon P \rightarrowtail X$ a homomorphism of coalgebras:*



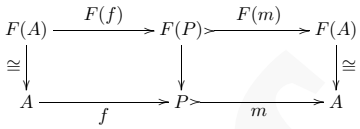
(ii) *Similarly, a predicate $m\colon P \rightarrowtail X$ is an invariant for an algebra $a\colon F(X) \to X$ if P carries a (necessarily unique) **subalgebra** structure $F(P) \to P$ making $m\colon P \rightarrowtail X$ a homomorphism of algebras.* ❑

Earlier we have seen a generic "binary" induction principle in Theorem 3.1.4. At this stage we can prove the familiar "unary" induction principle for initial algebras.

4.2.6. Theorem (Unary induction proof principle). *Every invariant on an initial algebra is always true.*

Equivalently, the truth predicate is the only invariant on an initial algebra. The proof is a generalisation of the argument we have used in Example 2.4.3 to derive induction for the natural numbers from initiality.

Proof. Assume $m\colon P \rightarrowtail A$ is an invariant on the initial algebra $F(A) \xrightarrow{\cong} A$. This means by the previous theorem that P itself carries a subalgebra structure $F(P) \to P$, making the square below on the right commute. This subalgebra yields a homomorphism $f\colon A \to P$ by initiality, as on the left:



By uniqueness we then get $m \circ f = \mathrm{id}_A$, which tells that $t \in P$, for all $t \in A$. ❑

4.2.7. Example. Consider the binary trees trees from Example 2.4.4 as algebras of the functor $T(X) = 1 + (X \times A \times X)$, with initial algebra

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

Predicate lifting $\mathrm{Pred}(T)(P) \subseteq T(X)$ of an arbitrary predicate $P \subseteq X$ is given by:

$$\mathrm{Pred}(T)(P) \quad = \quad \{\kappa_1(*)\} \cup \{\kappa_2(x_1, a, x_2) \mid a \in A \wedge P(x_1) \wedge P(x_2)\}.$$

Therefore, a predicate $P \subseteq \mathsf{BinTree}(A)$ on the initial algebra is an invariant if both:

$$\begin{cases} P(\mathsf{nil}) \\ P(x_1) \wedge P(x_2) \Rightarrow P(\mathsf{node}(x_1, a, x_2)) \end{cases}$$

The unary induction principle then says that such a P must hold for all binary trees $t \in \mathsf{BinTree}(A)$. This may be rephrased in rule form as:

$$\frac{P(\mathsf{nil}) \qquad P(x_1) \wedge P(x_2) \Rightarrow P(\mathsf{node}(x_1, a, x_2))}{P(t)}$$

4.2.1 Greatest invariants and limits of coalgebras

In the the first chapter, in Definition 1.3.2 to be precise, we already saw the predicate $\Box P$, describing "henceforth P" for a predicate P on sequences. Here we shall extend this same idea to arbitrary coalgebras. This will be used to prove an important result: existence of limits (products and equalisers) of coalgebras, see theorems 4.2.12 and 4.2.11. Along the same lines, greatest invariants can be used to construct right adjoints to functors between categories of coalgebras, see Exercise 4.2.5.

In Section 4.3 these greatest invariants will be studied more systematically in the context of temporal logic.

4.2.8. Definition. Let $c\colon X \to F(X)$ be a coalgebra of a polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$. For an arbitrary predicate $P \subseteq X$ on the state space of c, we define a new predicate $\Box P \subseteq X$, called **henceforth** P, as:

$$\big(\Box P\big)(x) \quad \text{iff} \quad \exists Q \subseteq X.\, Q \text{ is an invariant for } c \wedge Q \subseteq P \wedge Q(x).$$

Since invariants are closed under union—by Proposition 4.2.4 (i)—$\Box P$ is an invariant. Among all the invariants $Q \subseteq X$, it it the greatest one that is contained in P.

The definition of henceforth resembles the definition of bisimilarity (see Definition 3.1.5). In fact, one could push the similarity by defining for an arbitrary relation R, $\Box R$ as the greatest bisimilarity contained in R—so that bisimilarity $\leftrightarrows$ would appear as $\Box \top$. But there seems to be no clear use for this extra generality.

The next lemma lists some obvious properties of $\Box$. Some of these are already mentioned in Exercise 1.3.3 for sequences.

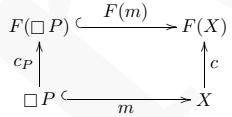
4.2.9. Lemma. *Consider the henceforth operator $\Box$ for a coalgebra $c\colon X \to F(X)$. The first three properties below express that $\Box$ is an interior operator. The fourth property says that its opens are invariants.*

(i) $\Box P \subseteq P$;
(ii) $\Box P \subseteq \Box\Box P$;
(iii) $P \subseteq Q \Rightarrow \Box P \subseteq \Box Q$;
(iv) *P is an invariant if and only if $P = \Box P$.*

Proof. (i) Obvious: if $\Box P(x)$, then $Q(x)$ for some invariant Q with $Q \subseteq P$. Hence $P(x)$.
(ii) If $\Box P(x)$, then we have an invariant Q, namely $\Box P$, with $Q(x)$ and $Q \subseteq \Box P$. Hence $\Box\Box P(x)$.
(iii) Obvious.
(iv) The (if)-part is clear because we have already seen that $\Box P$ is an invariant. For the (only if)-part, by (i) we only have to prove $P \subseteq \Box P$, if P is an invariant. So assume $P(x)$, then we have an invariant Q, namely P, with $Q(x)$ and $Q \subseteq P$. Hence $\Box P(x)$. ❑

The following result gives an important structural property of greatest invariants. It may be understood abstractly as providing a form of comprehension for coalgebras, as elaborated in Exercise 4.2.4 below.

4.2.10. Proposition. *Consider a coalgebra $c\colon X \to F(X)$ of a polynomial functor F with an arbitrary predicate $P \subseteq X$. By Theorem 4.2.5 (i) the greatest invariant $\Box P \subseteq P \subseteq X$ carries a subcoalgebra structure, say c_P, in:*



This subcoalgebra has the following universal property: each coalgebra homomorphism $f\colon \big(Y \xrightarrow{d} F(Y)\big) \longrightarrow \big(X \xrightarrow{c} F(X)\big)$ which factors through $P \subseteq X$—i.e. satisfies $f(y) \in P$ for all $y \in Y$—also factors through $\Box P$ as (unique) coalgebra homomorphism $f'\colon \big(Y \xrightarrow{d} F(Y)\big) \to \big(\Box P \xrightarrow{c_P} F(\Box P)\big)$ with $m \circ f' = f$.

Proof. The assumption that f factors through $P \subseteq X$ may be rephrased as an inclusion $\mathrm{Im}(f) \subseteq P$. But since the image along a homomorphism is an invariant, see Proposition 4.2.4 (ii), we get an inclusion $\mathrm{Im}(f) \subseteq \Box P$. This gives the factorisation:

$$\Big(Y \xrightarrow{\ f\ } X\Big) \;=\; \Big(Y \xrightarrow{\ f'\ } \Box P \xrightarrow{\ m\ } X\Big)$$

We only have to show that f' is a homomorphism of coalgebras. But this follows because $F(m)$ is injective, as noted in Lemma 4.1.4. It yields $c_P \circ f' = F(f') \circ d$ since:

$$\begin{aligned} F(m) \circ c_P \circ f' &= c \circ m \circ f' \\ &= c \circ f \\ &= F(f) \circ d \\ &= F(m) \circ F(f') \circ d. \end{aligned}$$

❑

In the remainder of this section we shall use greatest invariants to prove the the existence of limits equalisers and products of coalgebras. Recall from Exercises 2.1.14 and 2.1.13 that colimits (coequalisers and coproducts) are easy: they are constructed just like for sets. The product structure of coalgebras, however, is less trivial. First results appeared in [241], for "bounded" endofunctors on **Sets**. This was generalised in [105, 151, 124] and [134] (which is followed below). We begin with equalisers, which are easy using henceforth.

4.2.11. Theorem (Equalisers of coalgebras). *The category $\mathbf{CoAlg}(F)$ of coalgebras of a polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$ has equalisers: for two coalgebras $X \xrightarrow{c} F(X)$ and $Y \xrightarrow{d} F(Y)$ with homomorphisms $f, g\colon X \to Y$ between them there is an equaliser diagram in $\mathbf{CoAlg}(F)$,*

$$\begin{pmatrix} F(\Box \mathit{Equal}(f,g)) \\ \uparrow \\ \Box \mathit{Equal}(f,g) \end{pmatrix} \rightarrowtail \begin{pmatrix} F(X) \\ \uparrow c \\ X \end{pmatrix} \rightrightarrows \begin{pmatrix} F(Y) \\ \uparrow d \\ Y \end{pmatrix}$$

where $\mathit{Equal}(f,g) \hookrightarrow X$ is the equaliser $\{x \in X \mid f(x) = g(x)\}$ as in $\mathbf{Sets}$. The greatest invariant invariant $\Box \mathit{Equal}(f,g) \hookrightarrow \mathit{Equal}(f,g)$ carries a coalgebra structure by the previous proposition.

Proof. We show that the diagram above is universal in $\mathbf{CoAlg}(F)$: for each coalgebra $e\colon Z \to F(Z)$ with homomorphism $h\colon Z \to X$ satisfying $f \circ h = g \circ h$, the map h factors through $Z \to \mathrm{Equal}(f,g)$ via a unique function. By Proposition 4.2.10 h then restricts to a (unique) coalgebra homomorphism $Z \to \Box\,\mathrm{Equal}(f,g)$. ❑

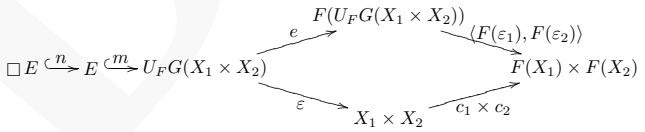
4.2.12. Theorem (Products of coalgebras). *For a polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$, the category $\mathbf{CoAlg}(F)$ of coalgebras has arbitrary products $\prod$.*

Proof. We shall construct the product of two coalgebras $c_i\colon X_i \to F(X_i)$, for $i = 1, 2$, and leave the general case to the reader. We first form the product $X_1 \times X_2$ of the underlying sets, and consider the cofree coalgebra on it, see Proposition 2.5.3. It will be written as $e\colon U_F G(X_1 \times X_2) \to F(U_F G(X_1 \times X_2))$, where $U_F\colon \mathbf{CoAlg}(F) \to \mathbf{Sets}$ is the forgetful functor, and G its right adjoint. It comes with a map $\varepsilon\colon U_F G(X_1 \times X_2) \to X_1 \times X_2$. For convenience we shall use the *ad hoc* notation $\varepsilon_i = \pi_i \circ \varepsilon\colon U_F G(X_1 \times X_2) \to X_i$.

Next we form the following equaliser (in $\mathbf{Sets}$).

$$E \;=\; \{u \in U_F G(X_1 \times X_2) \mid \forall i \in \{1,2\}.\, \big(c_i \circ \varepsilon_i\big)(u) = \big(F(\varepsilon_i) \circ e\big)(u)\}.$$

Then we take its greatest invariant $\Box E \subseteq E$ in the diagram describing E explicitly as equaliser:



By Proposition 4.2.10, the subset $\Box E \subseteq U_F G(X_1 \times X_2)$ carries an F-subcoalgebra structure, for which we write $c_1 \mathbin{\dot\times} c_2$ in:

$$\begin{array}{ccc} F(\Box E) & \xrightarrow{F(m \circ n)} & F(U_F G(X_1 \times X_2)) \\ {\scriptstyle c_1 \mathbin{\dot\times} c_2}\uparrow & & \uparrow{\scriptstyle e} \\ \Box E & \xrightarrow[m \circ n]{} & U_F G(X_1 \times X_2) \end{array} \tag{4.3}$$

The dot in $\mathbin{\dot\times}$ is used to distinguish this product of objects (coalgebras) from the product $c_1 \times c_2$ of functions, as used in the equaliser diagram above.

We claim this coalgebra $c_1 \mathbin{\dot\times} c_2\colon \Box E \to F(\Box E)$ is the product of the two coalgebras c_1 and c_2, in the category $\mathbf{CoAlg}(F)$. We thus follow the categorical description of product, from Definition 2.1.1. The two projection maps are

$$p_i \;\stackrel{\text{def}}{=}\; \varepsilon_i \circ m \circ n : \Box E \longrightarrow X_i.$$

We have to show that they are homomorphisms of coalgebras $c_1 \mathbin{\dot\times} c_2 \to c_i$. This follows from easy calculations:

$$\begin{array}{lcll} F(p_i) \circ (c_1 \mathbin{\dot\times} c_2) & = & F(\varepsilon_i) \circ F(m \circ n) \circ (c_1 \mathbin{\dot\times} c_2) & \\ & = & F(\varepsilon_i) \circ e \circ m \circ n & \text{see the above diagram (4.3)} \\ & = & \pi_i \circ (c_1 \times c_2) \circ \varepsilon \circ m \circ n & \text{since } m \text{ is an equaliser} \\ & = & c_i \circ \pi_i \circ \varepsilon \circ m \circ n & \\ & = & c_i \circ p_i. & \end{array}$$

Next we have to construct pairs, for coalgebra homomorphisms $f_i\colon \big(Y \xrightarrow{d} F(Y)\big) \longrightarrow \big(X_i \xrightarrow{c_i} F(X_i)\big)$. To start, we can form the pair $\langle f_1, f_2\rangle\colon Y \to X_1 \times X_2$ in $\mathbf{Sets}$. By cofreeness it gives rise to unique function $g\colon Y \to U_F G(X_1 \times X_2)$ forming a coalgebra homomorphism $d \to e$, with $\varepsilon \circ g = \langle f_1, f_2\rangle$. This g has the following equalising property.

$$\begin{array}{lcl} \langle F(\varepsilon_1), F(\varepsilon_2)\rangle \circ e \circ g & = & \langle F(\pi_1 \circ \varepsilon), F(\pi_2 \circ \varepsilon)\rangle \circ F(g) \circ d \\ & & \qquad \text{since } g\colon d \to e \\ & = & \langle F(\pi_1 \circ \langle f_1, f_2\rangle), F(\pi_2 \circ \langle f_1, f_2\rangle)\rangle \circ d \\ & = & \langle F(f_1) \circ d, F(f_2) \circ d\rangle \\ & = & \langle c_1 \circ f_1, c_2 \circ f_2\rangle \\ & & \qquad \text{because } f_i\colon d \to c_i \\ & = & \langle c_1 \circ \pi_1 \circ \varepsilon \circ g, c_2 \circ \pi_2 \circ \varepsilon \circ g\rangle \\ & = & (c_1 \times c_2) \circ \varepsilon \circ g. \end{array}$$

As a result, g factors through $m\colon E \hookrightarrow U_F G(X_1 \times X_2)$, say as $g = m \circ g'$. But then, by Proposition 4.2.10, g' also factors through $\Box E$. This yields the pair we seek: we write $\langle\langle f_1, f_2\rangle\rangle$ for the unique map $Y \to \Box E$ with $n \circ \langle\langle f_1, f_2\rangle\rangle = g'$.

We still have to show that this pair $\langle\langle f_1, f_2\rangle\rangle$ satisfies the required properties.

- The equations $p_i \circ \langle\langle f_1, f_2\rangle\rangle = f_i$ hold, since:

$$\begin{array}{lcl} p_i \circ \langle\langle f_1, f_2\rangle\rangle & = & \pi_i \circ \varepsilon \circ m \circ n \circ \langle\langle f_1, f_2\rangle\rangle \\ & = & \pi_i \circ \varepsilon \circ m \circ g' \\ & = & \pi_i \circ \varepsilon \circ g \\ & = & \pi_i \circ \langle f_1, f_2\rangle \\ & = & f_i. \end{array}$$

- The pair $\langle\langle f_1, f_2\rangle\rangle$ is the unique homomorphism with $p_i \circ \langle\langle f_1, f_2\rangle\rangle = f_i$. Indeed, if $h\colon Y \to \Box E$ is also a coalgebra map $d \to (c_1 \mathbin{\dot\times} c_2)$ with $p_i \circ h = f_i$, then $m \circ n \circ h$ is a coalgebra map $d \to e$ which satisfies:

$$\begin{array}{lcl} \varepsilon \circ m \circ n \circ h & = & \langle \pi_1 \circ \varepsilon \circ m \circ n \circ h, \pi_2 \circ \varepsilon \circ m \circ n \circ h\rangle \\ & = & \langle p_1 \circ h, p_2 \circ h\rangle \\ & = & \langle f_1, f_2\rangle. \end{array}$$

Hence $m \circ n \circ h = g$, and since $g = m \circ g'$ we get $n \circ h = g'$ because m is mono. Similarly, since $n \circ g' = \langle\langle f_1, f_2\rangle\rangle$ we get the required uniqueness: $h = \langle\langle f_1, f_2\rangle\rangle$. ❑

Since we have already seen that equalisers exist for coalgebras, we now know that all limits exist (see for instance [167, V,2]). Exercises 2.1.14 and 2.1.13 showed that colimits also exist. Hence we can summarise the situation as follows.

4.2.13. Corollary. *The category* $\mathbf{CoAlg}(F)$ *of coalgebras of a polynomial functor is both complete and cocomplete.* ❑

Exercises

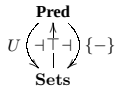
4.2.1. Let $F(X) \xrightarrow{a} X$ and $F(Y) \xrightarrow{b} Y$ be algebras of a polynomial functor F. Prove, in analogy with Lemma 4.2.2 that:
(i) If $R \subseteq X \times Y$ is a congruence, then both its domain $\coprod_{\pi_1} R \subseteq X$ and its codomain $\coprod_{\pi_2} R \subseteq Y$ are invariants.
(ii) If $P \subseteq X$ is an invariant, then $\coprod_\delta P \subseteq X \times X$ is a congruence.

4.2.2. Use binary induction in Theorem 3.1.4 together with the previous exericse to give an alternative proof of unary induction from Theorem 4.2.6.

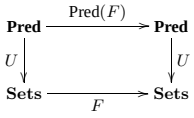
4.2.3. The next result from [85] is the analogue of Exercise 3.2.7; it describes when a function is definable by coinduction.
Let $Z \xrightarrow{\cong} F(Z)$ be final coalgebra of a polynomial functor F. Prove that an arbitrary function $f\colon X \to Z$ is defined by finality (*i.e.* is beh for some coalgebra $c\colon X \to F(X)$ on its domain X) if and only if its image $\mathrm{Im}(f) \subseteq Z$ is an invariant.

[*Hint.* Recall the following version of the Axiom of Choice: every surjective function $g\colon A \twoheadrightarrow B$ has a section, *i.e.* a function $s\colon B \to A$ with $g \circ s = \mathrm{id}_B$. Apply this to the surjective part $X \twoheadrightarrow \mathrm{Im}(f) \rightarrowtail Z$ in the factorisation of $f\colon X \to Z$.]

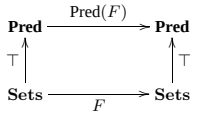
4.2.4. This exercise will first describe (ordinary) comprehension, or subset types see [130, 4.6], for sets in abstract, categorical form, and then show that the henceforth operator $\Box$ fits into this structure. It follows [134].
(i) Let **Pred** be the category of predicates $(P \subseteq X)$ on sets. Its morphisms $(P \subseteq X) \xrightarrow{f} (Q \subseteq Y)$ are functions $f\colon X \to Y$ mapping P to Q: $f(x) \in Q$ for all $x \in P$. Equivalently, $\coprod_f(P) \subseteq Q$ or $P \subseteq f^{-1}(Q)$. There is then an obvious forgetful functor $U\colon \mathbf{Pred} \to \mathbf{Sets}$ mapping $(P \subseteq X)$ to X. It is a "fibration", see [130], but that is not very relevant here.
There is a "truth" predicate functor $\top\colon \mathbf{Sets} \to \mathbf{Pred}$ sending a set X to the truth predicate $\top(X) = (X \subseteq X)$. Prove that $\top$ is right adjoint to U.
(ii) Define a "comprehension" or "subset type" functor $\{-\}\colon \mathbf{Pred} \to \mathbf{Sets}$ by $(P \subseteq X) \mapsto P$. Prove that $\{-\}$ is right adjoint to $\top$, so that there is a situation:

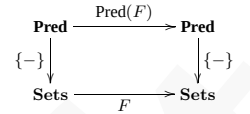


(iii) Let $F\colon \mathbf{Sets} \to \mathbf{Sets}$ now be a polynomial functor. Show that predicate lifting $\mathrm{Pred}(F)$ forms a functor in a commuting diagram:

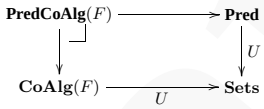


And prove that also the following two diagrams commute (see Lemma 4.1.3 (i) and Lemma 4.1.4).





(iv) We define a category **PredCoAlg**(F) of "predicates on coalgebras". Its objects are coalgebras-predicate pairs $\langle X \to F(X), P \subseteq X\rangle$. And its morphisms $\langle X \to F(X), P \subseteq X\rangle \longrightarrow \langle Y \to F(Y), Q \subseteq Y\rangle$ are coalgebra homomorphisms $f\colon (X \to F(X)) \longrightarrow (Y \to F(Y))$ which are at the same time morphisms of predicates: $P \subseteq f^{-1}(Q)$.
Check that this category **PredCoAlg**(F) can be formed as a pullback of categories:

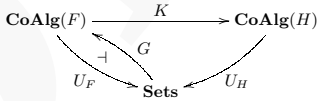


(v) Define a truth predicate functor $\top\colon \mathbf{CoAlg}(F) \to$ **PredCoAlg**(F) which is right adjoint to the forgetful functor **PredCoAlg**$(F) \to \mathbf{CoAlg}(F)$.
(vi) Prove that $\top$ has a right adjoint $\{-\}\colon$ **PredCoAlg**$(F) \to \mathbf{CoAlg}(F)$ given by:

$$\langle X \xrightarrow{c} F(X), P \subseteq X\rangle \longmapsto \Big(\Box P \xrightarrow{c_P} F(\Box P)\Big)$$

using the induced coalgebra c_P on the greatest invariant $\Box P$ from Proposition 4.2.10.

4.2.5. The next categorical result is a mild generalisation of [216, Theorem 17.1]. It involves an arbitrary functor K between categories of coalgebras, instead of a special functor induced by a natural transformation, like in Proposition 4.1.8. Also the proof hint that we give lead to a slightly more elementary proof than in [216] because it avoids bisimilarity and uses an equaliser (in $\mathbf{Sets}$) instead, much like in the proof of Theorem 4.2.12.
Consider two polynomial functors $F, H\colon \mathbf{Sets} \to \mathbf{Sets}$. Assume that there is a functor K between categories of coalgebras, commuting with the corresponding forgetful functors U_F and U_H, like in:



Prove that if F has cofree coalgebras, given by a right adjoint G to the forgetful functor U_F as in the diagram (and like in Proposition 2.5.3), then K has a right adjoint.
[*Hint.* For an arbitrary H-coalgebra $d\colon Y \to H(Y)$, first consider the cofree F-coalgebra on Y, say $e\colon U_F G(Y) \to F(U_F G(Y))$, and then form the equaliser

$$E \;=\; \{u \in U_F G(Y) \mid (K(e) \circ H(\varepsilon_Y))(u) = (d \circ \varepsilon_Y)(u)\}.$$

The greatest invariant $\Box E$ is then the carrier of the required F-coalgebra.]

4.3 Temporal logic of coalgebras

Modal logic is a branch of logic in which the notions of necessity and possibility are investigated, via special modal operators. It has developed into a field in which other notions like time, knowledge, program execution and provability are analysed in comparable manners, see for instance [69, 95, 122, 160, 226, 43]. The use of temporal logic for reasoning

about (reactive) state-based systems is advocated especially in [200, 201, 174], concentrating on temporal operators for transition systems—which may be seen as special instances of coalgebras (see Subsection 2.2.4). The coalgebraic approach to temporal logic extends these operators from transition systems to other coalgebras, in a uniform manner, following ideas first put forward by Moss [183] and subsequently developed by several others [209, 211, 165, 136, 190, 163]. This will be the subject of the present section.

We have already seen a few constructions with predicate lifting and invariants. Here we will elaborate the logical aspects, and will in particular illustrate how a tailor-made temporal logic can be associated with a coalgebra, via a generic definition. This follows [136]. The exposition starts with "forward" temporal operators, talking about future states, and will continue with "backward" operators in Subsection 4.3.1.

The logic in this section will deal with predicates on the state spaces of coalgebras. We extend the usual boolean connectives to predicates, via pointwise definitions: for $P, Q \subseteq X$,

$$\begin{array}{rcl} \neg P & = & \{x \in X \mid \neg P(x)\} \\ P \wedge Q & = & \{x \in X \mid P(x) \wedge Q(x)\} \\ P \Rightarrow Q & = & \{x \in X \mid P(x) \Rightarrow Q(x)\} \quad \textit{etc.} \end{array}$$

In Section 1.3 we have described a nexttime operator $\bigcirc$ for sequences. We start by generalising it to other coalgebras. This $\bigcirc$ will be used to construct more temporal operators.

4.3.1. Definition. Let $c\colon X \to F(X)$ be a coalgebra of a polynomial functor F. We define the **nexttime** operator $\bigcirc\colon \mathcal{P}(X) \to \mathcal{P}(X)$ as:

$$\begin{array}{rcl} \bigcirc P & = & c^{-1}\big(\mathrm{Pred}(F)(P)\big) \\ & = & \{x \in X \mid c(x) \in \mathrm{Pred}(F)(P)\}. \end{array}$$

We understand the predicate $\bigcirc P$ as true for those states x, all of whose immediate successor states, if any, satisfy P. This will be made precise in Proposition 4.3.7 below. Notice that we leave the dependence of the operator $\bigcirc$ on the coalgebra c (and the functor) implicit. Usually, this does not lead to confusion.

Here are some obvious results.

4.3.2. Lemma. *The above nexttime operator $\bigcirc$ satisfies the following properties.*

(i) *It is monotone: $P \subseteq Q \Rightarrow \bigcirc P \subseteq \bigcirc Q$. Hence it is an endofunctor $\mathcal{P}(X) \to \mathcal{P}(X)$ on the poset category of predicates ordered by inclusion.*

(ii) *It commutes with inverse images: $\bigcirc(f^{-1}Q) = f^{-1}(\bigcirc Q)$.*

(iii) *It has invariants as its coalgebras: $P \subseteq X$ is an invariant if and only if $P \subseteq \bigcirc P$.*

(iv) *It preserves meets (intersections) of predicates.*

(v) *The greatest invariant $\Box P$ from Definition 4.2.8 is the "cofree $\bigcirc$-coalgebra" on P: it is the final coalgebra—or greatest fixed point—of the operator $S \mapsto P \wedge \bigcirc S$ on $\mathcal{P}(X)$.*

Proof. We only illustrate the second and the last point. For a homomorphism of coalgebras $(X \xrightarrow{c} FX) \xrightarrow{f} (Y \xrightarrow{d} FY)$ and a predicate $Q \subseteq Y$ we have:

$$\begin{array}{rcll} \bigcirc(f^{-1}Q) & = & c^{-1}\mathrm{Pred}(F)(f^{-1}Q) & \\ & = & c^{-1}F(f)^{-1}\mathrm{Pred}(F)(Q) & \text{by Lemma 4.1.3 (ii)} \\ & = & f^{-1}d^{-1}\mathrm{Pred}(F)(Q) & \text{since } f \text{ is a homomorphism} \\ & = & f^{-1}(\bigcirc Q). & \end{array}$$

For the last point, first note that $\Box P$ is a coalgebra of the functor $P \wedge \bigcirc(-)$ on $\mathcal{P}(X)$. Indeed, $\Box P \subseteq P \wedge \bigcirc(\Box P)$, because $\Box P$ is contained in P and is an invariant. Next,

Notation	Meaning	Definition
$\bigcirc P$	nexttime P	$c^{-1}\mathrm{Pred}(F)(P)$
$\Box P$	henceforth P	$\nu S.\,(P \wedge \bigcirc S)$
$\Diamond P$	eventually P	$\neg\Box\neg P$
$P\,\mathcal{U}\,Q$	P until Q	$\mu S.\,(Q \vee (P \wedge \neg\bigcirc\neg S))$

Figure 4.1: Standard (forward) temporal operators.

$\Box P$ is the greatest such coalgebra, and hence the final one: if $Q \subseteq P \wedge \bigcirc Q$, then Q is an invariant contained in P, so that $Q \subseteq \Box P$. We conclude that $\Box P$ is the cofree $\bigcirc(-)$-coalgebra. ❑

The nexttime operator $\bigcirc$ is fundamental in temporal logic. By combining it with negations, least fixed points μ, and greatest fixed points ν one can define other temporal operators. For instance $\neg\bigcirc\neg$ is the so-called **strong nexttime** operator. It holds for those states for which there actually is a successor state satisfying P. The table in Figure 4.3 mentions a few standard operators.

We shall next illustrate the temporal logic of coalgebras in two examples.

4.3.3. Example. Douglas Hofstadter explains in his book *Gödel, Escher, Bach* [120] the object- and meta-level perspective on formal systems using a simple "MU-puzzle". It consists of a simple "Post" production system (see *e.g.* [62, Section 5.1]) or rewriting system for generating certain strings containing the symbols M, I, U. The meta-question that is considered is wheter the string MU can be produced. Both this production system and this question (and also its answer) can be (re)formulated in coalgebraic terminology.

Let therefore our alphabet A be the set $\{\mathsf{M}, \mathsf{I}, \mathsf{U}\}$ of relevant symbols. We will describe the production system as an unlabeled transition system (UTS) $A^\star \to \mathcal{P}_{\mathrm{fin}}(A^\star)$ on the the set $A^\star$ of strings over this alphabet. This is given by the following transitions (from [120]), which are parametrised by strings $x, y \in A^\star$.

$$x\mathsf{I} \longrightarrow x\mathsf{IU} \qquad \mathsf{M}x \longrightarrow \mathsf{M}xx \qquad x\mathsf{III}y \longrightarrow x\mathsf{U}y \qquad x\mathsf{UU}y \longrightarrow xy.$$

Thus, the associated transition system $A^\star \to \mathcal{P}_{\mathrm{fin}}(A^\star)$ is given by:

$$\begin{array}{rcl} w & \longmapsto & \{z \in A^\star \mid \exists x \in A^\star.\ (w = x\mathsf{I} \wedge z = x\mathsf{IU}) \\ & & \qquad\qquad \vee\ (w = \mathsf{M}x \wedge z = \mathsf{M}xx) \\ & & \qquad \vee \exists x, y \in A^\star.\ (w = x\mathsf{III}y \wedge z = x\mathsf{U}y) \\ & & \qquad\qquad \vee\ (w = x\mathsf{UU}y \wedge z = xy)\} \end{array}$$

It is not hard to see that for each w this set on the right-hand-side is finite.

The question considered in [120] is whether the string MU can be obtained from MI. That is, wheter $\mathsf{MI} \longrightarrow^* \mathsf{MU}$. Or, to put it into temporal terminology, whether the predicate

"equal to MU" eventually holds, starting from MI:

$$\begin{aligned}
&\Diamond(\{x \in A^\star \mid x = \mathsf{MU}\})(\mathsf{MI}) \\
&\iff \neg\Box(\neg\{x \in A^\star \mid x = \mathsf{MU}\})(\mathsf{MI}) \\
&\iff \neg\exists P \text{ invariant. } P \subseteq \neg\{x \in A^\star \mid x = \mathsf{MU}\} \wedge P(\mathsf{MI}) \\
&\iff \forall P \text{ invariant. } \neg\big(\forall x \in P.\, x \neq \mathsf{MU}) \wedge P(\mathsf{MI})\big) \\
&\iff \forall P \text{ invariant. } P(\mathsf{MI}) \;\Rightarrow\; \exists x \in P.\, x = \mathsf{MU} \\
&\iff \forall P \text{ invariant. } P(\mathsf{MI}) \;\Rightarrow\; P(\mathsf{MU}).
\end{aligned}$$

Hofstadter [120] provides a counter example, by producing an invariant P for which $P(\mathsf{MI})$, but not $P(\mathsf{MU})$, namely:

$$P(x) \quad \stackrel{\text{def}}{\iff} \quad \text{the number of I's in } x \text{ is not a multiple of 3.}$$

This P is clearly an invariant: of the above four parametrised transitions, the first and last one do not change the number of I's; in the second transition $\mathsf{M}x \longrightarrow \mathsf{M}xx$, if the number of I's in the right-hand-side, *i.e.* in xx, is $3n$, then n must be even, so that the number of I's in x (and hence in $\mathsf{M}x$) must already be a multiple of 3; a similar argument applies to the third transition. Thus, property P is an invariant. Once we have reached this stage we have P as counter example: clearly $P(\mathsf{MI})$, but not $P(\mathsf{MU})$. Thus MU cannot be obtained from MI.

This proof is essentially the same proof that Hofstadter provides, but of course he does not use the same coalgebraic formulation and terminology. However, he does call the property P 'hereditary'.

This concludes the example. The relation we have used between $\longrightarrow^*$ and $\Diamond$ will be investigated more systematically below, see especially in Proposition 4.3.7.

Here is another, more technical, illustration.

4.3.4. Example. This example assumes some familiarity with the untyped lambda-calculus, and especially with its theory of Böhm trees, see [28, Chapter 10]. It involves an operational model for head normal form reduction, consisting of a final coalgebra of certain trees. Temporal logic will be used to define an appropriate notion of "free variable" on these trees.

We fix a set V, and think of its elements as variables. We consider the polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$ given by

$$F(X) = 1 + \big(V^\star \times V \times X^\star\big) \tag{4.4}$$

In this example we shall often omit the coprojections κ_i and simply write $*$ for $\kappa_1(*) \in 1 + (V^\star \times V \times X^\star)$ and $(\vec{v}, w, \vec{x})$ for $\kappa_2(\vec{v}, w, \vec{x}) \in 1 + (V^\star \times V \times X^\star)$. Also, we shall write $\zeta\colon \mathcal{B} \xrightarrow{\cong} F(\mathcal{B})$ for the final F-coalgebra—which exists by Theorem 2.3.9.

Lambda terms are obtained from variables $x \in V$, application MN of two λ-terms M, N, and abstraction $\lambda x.\, M$. By an easy induction one then sees that an arbitrary term can be written of the form $\lambda x_1 \ldots x_n.\, yM_1 \ldots M_m$. The set Λ of λ-terms thus carries an F-coalgebra structure, given by the head-normal-form function $\mathsf{hnf}\colon \Lambda \to F(\Lambda)$, see [28, Section 8.3]: for $M \in \Lambda$,

$$\mathsf{hnf}(M) \;=\; \begin{cases} * & \text{if } M \text{ has no head normal form} \\ (\langle x_1, \ldots, x_n\rangle, y, \langle M_1, \ldots, M_m\rangle) & \text{if } M \text{ has head normal form} \\ & \lambda x_1 \ldots x_1.\, yM_1 \ldots M_m \end{cases}$$

We can now define the Böhm tree $\mathsf{BT}(M)$ of a λ-term M via finality:

$$\begin{array}{ccc} 1 + (V^\star \times V \times \Lambda^\star) & \xrightarrow{\;\mathrm{id} + (\mathrm{id} \times \mathrm{id} \times \mathsf{BT}^\star)\;} & 1 + (V^\star \times V \times \mathcal{B}^\star) \\ {\scriptstyle\mathsf{hnf}}\uparrow & & \cong\uparrow{\scriptstyle\zeta} \\ \Lambda & \xrightarrow[\mathsf{BT}]{} & \mathcal{B} \end{array}$$

We call the elements of $\mathcal{B}$ (abstract[1]) Böhm trees. We do not really need to know what these elements look like, because we can work with the universal property of $\mathcal{B}$, namely finality. But a picture may be useful. For $A \in \mathcal{B}$ we can write:

$$\zeta(A) = \bot \qquad \text{or} \qquad \zeta(A) = \left(\begin{array}{ccc} & \lambda x_1 \ldots x_n.\, y & \\ \diagup & & \diagdown \\ \zeta(A_1) & \cdots & \zeta(A_m) \end{array}\right)$$

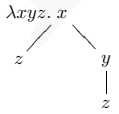
where the second picture applies when $\zeta(A) = (\langle x_1, \ldots, x_n\rangle, y, \langle A_1, \ldots, A_m\rangle)$. The '$\lambda$' is just syntactic sugar, used to suggest the analogy with the standard notation for Böhm trees [28]. The elements of $\mathcal{B}$ are thus finitely branching, possibly infinite, rooted trees, with labels of the form $\lambda x_1 \ldots x_n.\, y$, for variables $x_i, y \in V$.

Using the inverse $\zeta^{-1}\colon 1 + (\mathcal{B}^\star \times V \times \mathcal{B}^\star) \to \mathcal{B}$ of the final coalgebra we can explicitly construct Böhm trees. We give a few examples.

- Let us write $\bot_{\mathcal{B}} \in \mathcal{B}$ for $\zeta^{-1}(*)$. This the "empty" Böhm tree.
- The Böhm tree $\lambda x.\, x$ is obtained as $\zeta^{-1}(\langle x\rangle, x, \langle\rangle)$. In a similar way one can construct various kind of finite Böhm trees. For instance, the $\mathbf{S}$ combinator $\lambda xyz.\, xz(yz)$ is obtained as:

$$\zeta^{-1}(\langle x, y, z\rangle, x, \langle\zeta^{-1}(\langle\rangle, z, \langle\rangle), \zeta^{-1}(\langle\rangle, y, \langle\zeta^{-1}(\langle\rangle, z, \langle\rangle)\rangle)\rangle).$$

  Its picture is:



- Given an arbitrary Böhm tree $A \in \mathcal{B}$, we can define a new tree $\lambda x.\, A \in \mathcal{B}$ via λ-abstraction:

$$\lambda x.\, A \;=\; \begin{cases} \bot_{\mathcal{B}} & \text{if } \zeta(A) = * \\ \zeta^{-1}(x \cdot \vec{y}, z, \vec{B}) & \text{if } \zeta(A) = (\vec{y}, z, \vec{B}). \end{cases}$$

We proceed by using temporal logic to define free variables for Böhm trees. This requires some preliminary definitions. Let $x \in V$ be an arbitrary variable. It will be used

[1] One may have a more restricted view and call "Böhm tree" only those elements in $\mathcal{B}$ which actullay come from λ-terms, *i.e.* which are in the image of the function $\mathsf{BT}\colon \Lambda \to \mathcal{B}$. Then one may wish to call the elements of the whole set $\mathcal{B}$ "abstract" Böhm trees. We shall not do so. But it is good to keep in mind that the function BT is not surjective. For example, Böhm trees coming from λ-terms can only have a finite number of free variables (as defined below), whereas elements of $\mathcal{B}$ can have arbitrarily many.

in the auxiliary predicates Abs_x and Hv_x on Böhm trees, which are defined as follows: for $B \in \mathcal{B}$,

$$\begin{array}{rcl} \mathsf{Abs}_x(B) & \iff & \exists x_1, \ldots, x_n. \exists B_1, \ldots, B_m. \\ & & \quad B = \lambda x_1 \ldots x_n.\, y B_1 \ldots B_m \text{ and } x = x_i \text{ for some } i \\ \mathsf{Hv}_x(B) & \iff & \exists x_1, \ldots, x_n. \exists B_1, \ldots, B_m. \\ & & \quad B = \lambda x_1 \ldots x_n.\, y B_1 \ldots B_m \text{ and } x = y. \end{array}$$

Thus Abs_x describes the occurrence of x in the abstracted variables, and Hv_x that x is the head variable.

For a Böhm tree $A \in \mathcal{B}$ we can now define the set $\mathsf{FV}(A) \subseteq V$ of *free variables* in A via the until operator $\mathcal{U}$ from Figure 4.3:

$$x \in \mathsf{FV}(A) \iff \big(\neg \mathsf{Abs}_x \,\mathcal{U}\, (\mathsf{Hv}_x \wedge \neg \mathsf{Abs}_x)\big)(A).$$

In words: a variable x is free in a Böhm tree A if there is a successor tree B of A in which x occurs as "head variable", and in all successor trees of A until that tree B is reached, including B itself, x is not used in a lambda abstraction. This until formula then defines a predicate on $\mathcal{B}$, namely '$x \in \mathsf{FV}(-)$'.

There are then two crucial properties that we would like to hold for a Böhm tree A.

1. If $A = \bot_{\mathcal{B}}$, then
$$\mathsf{FV}(A) \;=\; \emptyset.$$
This holds because if $A = \bot_{\mathcal{B}}$, then both $\mathsf{Abs}_x(A)$ and $\mathsf{Hv}_x(A)$ are false, so that the least fixed point in Figure 4.3 defining $\mathcal{U}$ at A in $x \in \mathsf{FV}(A)$ is $\mu S.\, \neg \bigcirc \neg S$. This yields the empty set.

2. If $A = \lambda x_1 \ldots x_n.\, y A_1 \ldots A_m$, then
$$\mathsf{FV}(A) \;=\; \big(\{y\} \cup \mathsf{FV}(A_1) \cup \cdots \cup \mathsf{FV}(A_m)\big) - \{x_1, \ldots, x_n\}.$$
This result follows from the fixed point property (indicated as 'f.p.' below) defining the until operator $\mathcal{U}$ in Figure 4.3:
$$\begin{array}{rcl} x \in \mathsf{FV}(A) & \overset{\text{def}}{\iff} & \big[\neg \mathsf{Abs}_x \,\mathcal{U}\, (\mathsf{Hv}_x \wedge \neg \mathsf{Abs}_x)\big](A) \\ & \overset{\text{f.p.}}{\iff} & \big[(\mathsf{Hv}_x \wedge \neg \mathsf{Abs}_x) \vee (\neg \mathsf{Abs}_x \wedge \neg \bigcirc \neg (x \in \mathsf{FV}(-)))\big](A) \\ & \iff & \neg \mathsf{Abs}_x(A) \wedge \big(\mathsf{Hv}_x(A) \vee \neg \bigcirc \neg (x \in \mathsf{FV}(-))(A)\big) \\ & \iff & x \notin \{x_1, \ldots, x_n\} \wedge \big(x = y \vee \exists j \leq m.\, x \in \mathsf{FV}(A_j)\big) \\ & \iff & x \in \big(\{y\} \cup \mathsf{FV}(A_1) \cup \cdots \cup \mathsf{FV}(A_m)\big) - \{x_1, \ldots, x_n\}. \end{array}$$

This shows how temporal operators can be used to define sensible predicates on infinite structures. The generic definitions prove to provide adequate expressive power in concrete situations. We should emphasise however that the final coalgebra $\mathcal{B}$ of Böhm trees is only an operational model of the lambda calculus and not a denotational one: for instance, it is not clear how to define an application operation $\mathcal{B} \times \mathcal{B} \to \mathcal{B}$ on our abstract Böhm trees via coinduction. Such application is defined on the Böhm model used in [28, Section 18.3] via finite approximations. For more information on models of the (untyped) λ-calculus, see *e.g.* [28, Part V], [130, Section 2.5], or [71].

Our next application of temporal logic does not involve a specific functor, like for Böhm trees above, but is generic. It involves an (unlabeled) transition relation for an arbitrary coalgebra. Before we give the definition it is useful to introduce some special notation, and some associated results.

4.3.5. Lemma. *For an arbitrary set X and an element $x \in X$ we write introduce a "singleton" and "non-singleton" predicate on X:*

$$\begin{array}{rcl} (\cdot = x) & = & \{y \in X \mid y = x\} \\ & = & \{x\}. \\ (\cdot \neq x) & = & \{y \in X \mid y \neq x\} \\ & = & \neg(\cdot = x). \end{array}$$

Then:

(i) *For a predicate $P \subseteq X$,*
$$P \subseteq (\cdot \neq x) \iff \neg P(x).$$

(ii) *For a function $f\colon Y \to X$,*
$$f^{-1}\big(\cdot \neq x\big) \;=\; \bigcap_{y \in f^{-1}(x)} (\cdot \neq y).$$

(iii) *And for a polynomial functor F and a predicate $Q \subseteq F(X)$,*
$$\underleftarrow{\mathit{Pred}}(F)(Q) \iff \{x \in X \mid Q \not\subseteq \mathit{Pred}(F)(\cdot \neq x)\}$$

where $\underleftarrow{\mathit{Pred}}(F)$ is the "predicate lowering" left adjoint to predicate lifting $\mathit{Pred}(F)$ from Subsection 4.1.1.

Proof. Points (i) + (ii) follow immediately from the definition. For (iii) we use (i) in:

$$\begin{array}{rcl} x \in \underleftarrow{\mathrm{Pred}}(F)(Q) & \iff & \underleftarrow{\mathrm{Pred}}(F)(Q) \not\subseteq (\cdot \neq x) \\ & \iff & Q \not\subseteq \mathrm{Pred}(F)(\cdot \neq x). \end{array}$$

❑

In Proposition 4.1.7 we have seen an abstract way to turn an arbitrary coalgebra into an unlabeled transition system. Here, and later on in Theorem 4.3.9, we shall reconsider this topic from a temporal perspective.

4.3.6. Definition. Assume we have a coalgebra $c\colon X \to F(X)$ of a polynomial functor F. On states $x, x' \in X$ we define a transition relation via the strong nexttime operator, as:

$$\begin{array}{rcl} x \longrightarrow x' & \iff & x \in \big(\neg \bigcirc \neg\big)(\cdot = x') \\ & \iff & x \notin \bigcirc(\cdot \neq x'). \end{array}$$

This says that there is a transition $x \longrightarrow x'$ if and only if there is successor state of x which is equal to x'. In this way we turn an arbitrary coalgebra into an unlabeled transition system.

We shall first investigate the properties of this new transition system $\longrightarrow$, and only later in Theorem 4.3.9 show that it is actually the same as the earlier translation from coalgebras to transition systems from Subsection 4.1.1.

So let us first consider what we get for a coalgebra $c\colon X \to \mathcal{P}(X)$ of the powerset functor. Then the notation $x \longrightarrow x'$ is standardly used for $x' \in c(x)$. This coincides with Definition 4.3.6 since:

$$\begin{array}{rcll} x \notin \bigcirc(\cdot \neq x') & \iff & x \notin c^{-1}\big(\mathrm{Pred}(\mathcal{P})(\cdot \neq x')\big) & \\ & \iff & c(x) \notin \{a \mid a \subseteq (\cdot \neq x')\} & \\ & \iff & c(x) \notin \{a \mid x' \notin a\}, & \text{by Lemma 4.3.5 (i)} \\ & \iff & x' \in c(x). & \end{array}$$

Now that we have gained some confidence in this temporal transition definition, we consider further properties. It turns out that the temporal operators can be expressed in terms of the new transition relation.

4.3.7. Proposition. *The transition relation $\longrightarrow$ from Definition 4.3.6, induced by a coalgebra $X \to F(X)$, and its reflexive transitive closure $\longrightarrow^*$, satisfy the following properties.*

(i) For a predicate $P \subseteq X$,

(a) $\bigcirc P = \{x \in X \mid \forall x'.\, x \longrightarrow x' \;\Rightarrow\; P(x')\}$
(b) $\Box P = \{x \in X \mid \forall x'.\, x \longrightarrow^* x' \;\Rightarrow\; P(x')\}$
(c) $\Diamond P = \{x \in X \mid \exists x'.\, x \longrightarrow^* x' \wedge P(x')\}$.

This says that the temporal operators on the original coalgebra are the same as the ones on the induced unlabeled transition system.

(ii) For a predicate $P \subseteq X$, the following three statements are equivalent.

(a) P is an invariant;
(b) $\forall x, x' \in X.\, P(x) \wedge x \longrightarrow x' \;\Rightarrow\; P(x')$*;*
(c) $\forall x, x' \in X.\, P(x) \wedge x \longrightarrow^* x' \;\Rightarrow\; P(x')$.

(iii) For arbitrary states $x, x' \in X$, the following are equivalent.

(a) $x \longrightarrow^* x'$*;*
(b) $P(x) \Rightarrow P(x')$*, for all invariants $P \subseteq X$;*
(c) $x \in \Diamond(\cdot = x')$*, i.e. eventually there is successor state of x which is equal to x'.*

Proof. (i) We reason as follows.

$$\begin{array}{lll}
x \in \bigcirc P & \iff & c(x) \in \mathsf{Pred}(F)(P) \\
& \iff & \{c(x)\} \subseteq \mathsf{Pred}(F)(P) \\
& \iff & \underleftarrow{\mathsf{Pred}}(F)(\{c(x)\}) \subseteq P \\
& \iff & \forall x'.\, x' \in \underleftarrow{\mathsf{Pred}}(F)(\{c(x)\}) \Rightarrow P(x') \\
& \iff & \forall x'.\, \{c(x)\} \not\subseteq \mathsf{Pred}(F)(\cdot \neq x') \Rightarrow P(x'), \quad \text{by Lemma 4.3.5 (iii)} \\
& \iff & \forall x'.\, c(x) \notin \mathsf{Pred}(F)(\cdot \neq x') \Rightarrow P(x') \\
& \iff & \forall x'.\, x \longrightarrow x' \Rightarrow P(x').
\end{array}$$

For the inclusion ($\subseteq$) of (b) we can use (a) an appropriate number of times since $\Box P \subseteq \bigcirc \Box P$ and $\Box P \subseteq P$. For ($\supseteq$) we use that the predicate $\{x \mid \forall x'.\, x \longrightarrow^* x' \;\Rightarrow\; P(x')\}$ contains P and is an invariant, via (a); hence it is contained in $\Box P$.

The third point (c) follows directly from (b) since $\Diamond = \neg \Box \neg$.

(ii) Immediate from (i) since P is an invariant if and only if $P \subseteq \bigcirc P$, if and only if $P \subseteq \Box P$.

(iii) The equivalence (b) $\Leftrightarrow$ (c) follows by unfolding the definitions. The implication (a) $\Rightarrow$ (b) follows directly from (ii), but for the reverse we have to do a bit of work. Assume $P(x) \Rightarrow P(x')$ for all invariants P. In order to prove $x \longrightarrow^* x'$, consider the predicate $Q(y) \stackrel{\text{def}}{\iff} x \longrightarrow^* y$. Clearly $Q(x)$, so $Q(x')$ follows once we have established that Q is an invariant. But this is an easy consequence using (ii): if $Q(y)$, *i.e.* $x \longrightarrow^* y$, and $y \longrightarrow y'$, then clearly $x \longrightarrow^* y'$, which is $Q(y')$. ❑

4.3.1 Backward reasoning

So far in this section we have concentrated on "forward" reasoning, by only considering operators that talk about future states. However, within the setting of coalgebras there is also a natural way to reason about previous states. This happens via predicate lowering instead of via predicate lifting, *i.e.* via the left adjoint $\underleftarrow{\mathsf{Pred}}(F)$ to $\mathsf{Pred}(F)$, introduced in Subsection 4.1.1.

It turns out that the forward temporal operators have backward counterparts. We shall use notation with backwards underarrows for these analogues: $\underleftarrow{\bigcirc}$, $\underleftarrow{\Box}$ and $\underleftarrow{\Diamond}$ are backward versions of $\bigcirc$, $\Box$ and $\Diamond$.

4.3.8. Definition. For a coalgebra $c\colon X \to F(X)$ of a polynomial functor F, and a predicate $P \subseteq X$ on its carrier X, we define a new predicate **lasttime** P on X by

$$\begin{array}{rcl}
\underleftarrow{\bigcirc} P & = & \underleftarrow{\mathsf{Pred}}(F)(\coprod_c P) \\
& = & \underleftarrow{\mathsf{Pred}}(F)(\{c(x) \mid x \in P\}).
\end{array}$$

This is the so-called **strong lasttime** operator, which holds of a state x if there is an (immediate) predecessor state of x which satisfies P. The corresponding *weak* lasttime is $\neg \underleftarrow{\bigcirc} \neg$.

One can easily define an infinite extension of $\underleftarrow{\bigcirc}$, called **earlier**:

$$\begin{array}{rcl}
\underleftarrow{\Diamond} P & = & \text{"the least invariant containing } P\text{"} \\
& = & \{x \in X \mid \forall Q, \text{invariant. } P \subseteq Q \;\Rightarrow\; Q(x)\}.
\end{array}$$

This predicate $\underleftarrow{\Diamond} P$ holds of a state x if there is some (non-immediate) predecessor state of x for which P holds.

Figure 4.3.1 gives a brief overview of the main backward temporal operators. In the remainder of this section we shall concentrate on the relation between the backward temporal operators and transitions.

But first we give a result that was already announced. It states an equivalence between various (unlabeled) transition systems induced by coalgebras.

4.3.9. Theorem. *Consider a coalgebra $c\colon X \to F(X)$ of a polynomial functor F. Using the lasttime operator $\underleftarrow{\bigcirc}$ one can also define an unlabeled transition system by*

$$\begin{array}{rcl}
x \longrightarrow x' & \iff & \textit{"there is an immediate predecessor state of } x' \textit{ which is equal to } x\textit{"} \\
& \iff & x' \in \underleftarrow{\bigcirc}(\cdot = x).
\end{array}$$

This transition relation is then the same as
(i) $x \notin \bigcirc(\cdot \neq x')$ *from Definition 4.3.6;*
(ii) $x' \in \underleftarrow{\mathsf{Pred}}(F)(\{c(x)\})$ *used in the translation in (4.2), in Subsection 4.1.1.*

Proof. All these forward and backward transition definitions are equivalent because:

$$\begin{array}{lll}
x' \in \underleftarrow{\bigcirc}(\cdot = x) & \iff & x' \in \underleftarrow{\mathsf{Pred}}(F)(\coprod_c(\cdot = x)) \quad \text{by Definition 4.3.8} \\
& \iff & x' \in \underleftarrow{\mathsf{Pred}}(F)(\{c(x)\}) \quad \text{as used in (4.2)} \\
& \iff & \{c(x)\} \not\subseteq \mathsf{Pred}(F)(\cdot \neq x') \quad \text{by Lemma 4.3.5 (iii)} \\
& \iff & x \notin \bigcirc(\cdot \neq x') \quad \text{as used in Definition 4.3.6.}
\end{array}$$
❑

Notation	Meaning	Definition	Galois connection
$\bigcirc P$	lasttime P	$\underleftarrow{\mathrm{Pred}}(F)(\coprod_c P)$	$\bigcirc \dashv \bigcirc$
$\Diamond P$	(sometime) earlier P	$\mu S.\,(P \vee \bigcirc S)$	$\Diamond \dashv \Box$
$\Box P$	(always) before P	$\neg \Diamond \neg P$	$\Diamond \dashv \Box$
$P \mathcal{S} Q$	P since Q	$\mu S.\,(Q \vee (P \wedge \bigcirc S))$	

Figure 4.2: Standard (backward) temporal operators.

Finally we mention the descriptions of the backward temporal operators in terms of transitions, like in Proposition 4.3.7 (i).

4.3.10. Proposition. *For a predicate $P \subseteq X$ on the state space of a coalgebra,*

(i) $\bigcirc P = \{x \in X \mid \exists y.\, y \longrightarrow x \wedge P(y)\}$

(ii) $\Diamond P = \{x \in X \mid \exists y.\, y \longrightarrow^* x \wedge P(y)\}$

(iii) $\Box P = \{x \in X \mid \forall y.\, y \longrightarrow^* x \;\Rightarrow\; P(y)\}$.

Proof. Assume that $c\colon X \to F(X)$ is the coalgebra we are dealing with.

(i)
$$\begin{aligned} \bigcirc P &= \underleftarrow{\mathrm{Pred}}(F)(\{c(y) \mid y \in P\}) \\ &= \underleftarrow{\mathrm{Pred}}(F)(\textstyle\bigcup_{y\in P}\{c(y)\}) \\ &= \textstyle\bigcup_{y\in P} \underleftarrow{\mathrm{Pred}}(F)(\{c(y)\}) \qquad \text{since } \underleftarrow{\mathrm{Pred}}(F) \text{ is a left adjoint} \\ &= \{x \in X \mid \exists y \in P.\, x \in \textstyle\bigcirc(\cdot = y)\} \\ &= \{x \in X \mid \exists y.\, y \longrightarrow x \wedge P(y)\}. \end{aligned}$$

(ii) Let us write $P' = \{x \in X \mid \exists y.\, y \longrightarrow^* x \wedge P(y)\}$ for the right hand side. We have to prove that P' is the least invariant containing P.

- Clearly $P \subseteq P'$, by taking no transition.
- Also P' is an invariant, by Proposition 4.3.7 (ii): if $P'(x)$, say with $y \longrightarrow^* x$ where $P(y)$, and $x \longrightarrow x'$, then also $y \longrightarrow^* x'$ and thus $P'(x')$.
- If $Q \subseteq X$ is an invariant containing P, then $P' \subseteq Q$: if $P'(x)$, say with $y \longrightarrow^* x$ where $P(y)$; then $Q(y)$, and thus $Q(x)$ by Proposition 4.3.7 (ii).

(iii) Immediately from the definition $\Box = \neg \Diamond \neg$. ❑

Exercises

4.3.1. Consider the transition system $A^\star \to \mathcal{P}_{\mathrm{fin}}(A^\star)$ from Example 4.3.3, and prove:

$$\Box(\lambda x \in A^\star.\, \exists y \in A^\star.\, x = \mathsf{M}y)(\mathsf{MI}).$$

This property is also mentioned in [120].

4.3.2. Prove the following "induction rule of temporal logic":

$$P \wedge \Box(P \Rightarrow \bigcirc P) \quad \subseteq \quad \Box P.$$

[Aside: using the term 'induction' for a rule that follows from a greatest fi xed point property is not very fortunate.]

4.3.3. Prove that for a predicate P on the state space of coalgebra of a polynomial functor,

$$\Box P \;=\; \bigcap_{n\in\mathbb{N}} \bigcirc^n P \qquad \text{and} \qquad \Diamond P \;=\; \bigcup_{n\in\mathbb{N}} \bigcirc^n P.$$

(Where $\bigcirc^0 P = P$, and $\bigcirc^{n+1} P = \bigcirc \bigcirc^n P$, and similarly for $\bigcirc$.)

4.3.4. Prove that:

$$\Box(f^{-1}Q) = f^{-1}(\Box Q) \qquad \bigcirc(\textstyle\coprod_f P) = \coprod_f(\bigcirc P) \qquad \Diamond(\coprod_f P) = \coprod_f(\Diamond P)$$

when f is a homomorphism of coalgebras.

4.3.5. Consider the transition relation $\longrightarrow$ from Defi nition 4.3.6, and use Lemma 4.3.5 (ii) to prove that for a homorphism $f\colon X \to Y$ of coalgebras,

$$f(x) \longrightarrow y \iff \exists x'.\, x \longrightarrow x' \wedge f(x') = y.$$

Note that this states the functoriality of the translation from coalgebras to transitition systems like in (4.2).

4.3.6. Prove—and explain in words—that

$$x \longrightarrow^* x' \iff x' \in \Diamond(\cdot = x).$$

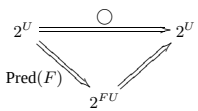
[The notation $\langle x\rangle = \{x' \mid x \longrightarrow^* x'\}$ and $\langle P\rangle = \{x' \mid \exists x \in P.\, x \longrightarrow^* x'\}$ is used in [216] for the least invariants $\Diamond\{x\} = \Diamond(\cdot = x)$ and $\Diamond P$ containing an element x or a predicate P.]

4.3.7. Verify the Galois connections in Figure 4.3.1.
[Such Galois connections for temporal logic are studied systematically in [154, 136].]

4.3.8. Check that $P \,\mathcal{U}\, P = P$.

4.3.9. The following is taken from [65, Section 5], where it is referred to as the Whisky Problem. It is used there as a challenge in proof automation in linear temporal logic. Here it will be formulated in the temporal logic of an arbitrary coalgebra (of a polynomial functor).
Consider an arbitrary set A with an endofunction $h\colon A \to A$. Let $P\colon A \to \mathcal{P}(X)$ be a parametrised predicate on the state space of a coalgebra X, satisfying for a specifi c $a \in A$ and $y \in X$:
(i) $P(a)(y)$;
(ii) $\forall b \in A.\, P(b)(y) \Rightarrow P(h(b))(y)$;
(iii) $\Box\Big(\{x \in X \mid \forall b \in A.\, P(h(b))(x) \Rightarrow \bigcirc(P(b))(x)\}\Big)(y)$.
Prove then that $\Box(P(a))(y)$.
[*Hint*. Use Exercise 4.3.3.]

4.3.10. Describe the nexttime operator $\bigcirc$ as a natural transformation $2^U \Rightarrow 2^U$, like in Exercise 4.1.3. Show that it can be described as a composition of natural transformations



4.3.11. Prove that the "until" and "since" operators $\mathcal{U}, \mathcal{S}\colon \mathcal{P}(X) \times \mathcal{P}(X) \to \mathcal{P}(X)$ on the state space X of a coalgebra (see Figures 4.3 and 4.3.1) can be described in the following way in terms of the transition relation $\longrightarrow \subseteq X \times X$ from Defi nition 4.3.6.

$$\begin{aligned} P \,\mathcal{U}\, Q &= \{x \mid \exists n.\, \exists x_0, \ldots, x_n.\, x_0 = x \wedge (\forall i < n.\, x_i \longrightarrow x_{i+1}) \wedge Q(x_n) \\ &\qquad \wedge \forall i < n.\, P(x_i)\} \\ P \,\mathcal{S}\, Q &= \{x \mid \exists n.\, \exists x_0, \ldots, x_n.\, x_n = x \wedge (\forall i < n.\, x_i \longrightarrow x_{i+1}) \wedge Q(x_0) \\ &\qquad \wedge \forall i > 0.\, P(x_i)\}. \end{aligned}$$

4.3.12. We consider the strong nexttime operator $\neg\bigcirc\neg$ associated with a coalgebra, and call a predicate P **maintainable** if $P \subseteq \neg\bigcirc\neg P$. Notice that such a predicate is a $\neg\bigcirc\neg$-coalgebra.

(i) Investigate what this requirement means, for instance for a few concrete coalgebras.

(ii) Let us use the notation $\mathsf{EA}\,P$ for the greatest maintainable predicate contained in P. Describe $\mathsf{EA}\,P$ in terms of the transition relation $\longrightarrow$ from Defi nition 4.3.6.

(iii) Similarly for $\mathsf{AE}\,P \stackrel{\text{def}}{=} \neg\mathsf{EA}\neg P$

[Operators like EA and AE are used in computation tree logic (CTL), see *e.g.* [69] to reason about paths in trees of computations. The interpretations we use here involve infi nite paths.]

4.4 Existence of fi nal coalgebras

At various places in this text final coalgebras have already been used. They have been described explicitly for a number of special functors, like in Proposition 2.3.5. Theorem 2.3.9 has mentioned that a final coalgebra exists for each finite polynomial functor. It is the main aim in this section to prove this result. The proofs are somewhat technical, and skipping the details of this section should not be problematic for following the rest of the text—except possibly the next section on trace semantics.

Actually, this section will describe two standard constructions, one for so-called ω-continuous[2] functors (including the simple polynomial functors), and one for ω-accessible functors (including the finite polynomial ones). The second construction is thus more general. But we include the first one as well because:

- it is a straightforward generalisation from well-known fixed point constructions in order theory;
- it allows one to give an intuitive description of initial algebras and final coalgebras;
- it will be used explicitly in the next section.

The second construction makes use of the temporal logic of coalgebras from the previous section. That it the reason why we have postponed this topic for so long.

Both these constructions will be used in the category of sets, although the mechanism works more generally. For more information we refer to the literature [225, 7, 29, 10, 242, 216, 100, 210].

4.4.1 Final coalgebras for ω-continuous functors

It is a well-known fact from order theory—see *e.g.* [64, Chapter 4]—that the least fixed point of a continuous function $f: X \to X$ on a suitable complete order X with a least element $\bot \in X$ can be computed by iteration, namely as join $\bigvee_{n\in\mathbb{N}} f^n(\bot)$. By dualisation, the greatest fixed point can be obtained as meet $\bigwedge_{n\in\mathbb{N}} f^n(\top)$—where $\top$ is the greatest element. This construction can be generalised to categories (as in [225, Lemma 2]), once the relevant notions have been suitably extended. This will be done first.

Consider an endofunctor $F: \mathbb{C} \to \mathbb{C}$ on a category $\mathbb{C}$ with a final object $1 \in \mathbb{C}$. Then we can consider the following "ω-chain"

$$1 \xleftarrow{\;!\;} F(1) \xleftarrow{F(!)} F^2(1) \xleftarrow{F^2(!)} F^3(1) \xleftarrow{F^3(!)} \cdots \tag{4.5}$$

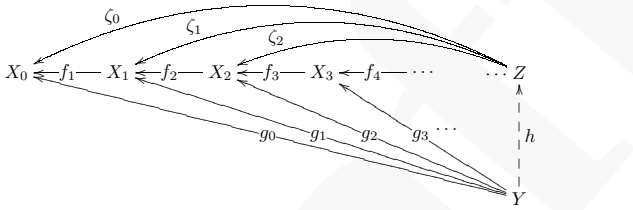
where ! is the unique map to the final object 1. If there is a "limit" Z of this chain, and if F then "preserves" this limit, one gets a suitable final coalgebra $Z \xrightarrow{\cong} F(Z)$. This generalises the order theoretic situation.

[2] The Greek letter ω is often used in mathematical logic for the set $\mathbb{N}$ of natual numbers. It is standard in this context.

We shall make this idea more precise. Therefore we consider a more general ω-chain in $\mathbb{C}$ of the form:

$$X_0 \xleftarrow{f_1} X_1 \xleftarrow{f_2} X_2 \xleftarrow{f_3} X_3 \xleftarrow{f_4} \cdots$$

A **limit** of such a chain—if it exists—is an object $Z \in \mathbb{C}$ with a collection of arrows $\big(Z \xrightarrow{\zeta_n} X_n\big)_{n\in\mathbb{N}}$ satisfying $f_{n+1} \circ \zeta_{n+1} = \zeta_n$, with the following universal property. For each object $Y \in \mathbb{C}$ with arrows $g_n: Y \to X_n$ such that $f_{n+1} \circ g_{n+1} = g_n$, there is a unique map $h: Y \to Z$ with $\zeta_n \circ h = g_n$, for each $n \in \mathbb{N}$. In a diagram:



A functor $F: \mathbb{C} \to \mathbb{D}$ is said to **preserve** such a limit if the chain $\big(F(X_n) \xrightarrow{F(\zeta_n)} F(Z)\big)_{n\in\mathbb{N}}$ resulting from applying F is a limit in the category $\mathbb{D}$. Another way to formulate this is: the induced map $F(Z) \to Z$ is an isomorphism. The functor F is called **ω-continuous** if it preserves limits of all ω-chains.

The main reason for considering these constructions is the following result.

4.4.1. Lemma. *Let $\mathbb{C}$ be a category with limits of ω-chains. Each continuous endofunctor $F: \mathbb{C} \to \mathbb{C}$ then has a final coalgebra, obtained as limit $Z \xrightarrow{\cong} F(Z)$ of the chain $\big(F^{n+1}(1) \xrightarrow{F^n(!)} F^n(1)\big)_{n\in\mathbb{N}}$ in (4.5).*

Proof. Applying F to the chain (4.5) and its limit Z yields another chain with limit $F(Z)$. Using the latter's universal property yields an isomorphism $\zeta: Z \xrightarrow{\cong} F(Z)$ with $F(\zeta_n) \circ \zeta = \zeta_{n+1}$. It is a final coalgebra, since for an arbitrary coalgebra $c: Y \to F(Y)$ we can form a collection of maps $c_n: Y \longrightarrow F^n(1)$ via:

$$\begin{array}{rcl} c_0 & = & \big(Y \xrightarrow{\;!\;} 1\big) \\ c_1 & = & \big(Y \xrightarrow{\;c\;} F(Y) \xrightarrow{F(!)} F(1)\big) \\ c_2 & = & \big(Y \xrightarrow{\;c\;} F(Y) \xrightarrow{F(c)} F^2(1) \xrightarrow{F^2(!)} F^2(1)\big) \\ & \vdots & \\ c_{n+1} & = & F(c_n) \circ c. \end{array}$$

The maps c_n commute with the arrows in the chain, which is easily seen by induction. This yields a unique map $h: Y \to Z$ with $\zeta_n \circ h = c_n$. It forms a homomorphism of coalgebras, *i.e.* satisfies $\zeta \circ h = F(h) \circ c$, by uniqueness of maps $Y \to F(Z)$ to the limit $F(Z)$:

$$\begin{array}{rcl} F(\zeta_n) \circ \zeta \circ h & = & \zeta_{n+1} \circ h \\ & = & c_{n+1} \\ & = & F(c_n) \circ c \\ & = & F(\zeta_n) \circ F(h) \circ c. \end{array}$$

❑

After these abstract considerations we show that they apply to simple polynomial functors on sets.

4.4.2. Lemma. (i) *In* $\mathbf{Sets}$ *limits of* ω*-chains exist, and are computed as follows. For a chain* $\big(X_{n+1} \xrightarrow{f_n} X_n\big)_{n\in\mathbb{N}}$ *the limit* Z *is a subset of the infinite product* $\prod_{n\in\mathbb{N}} X_n$ *given by:*

$$Z \;=\; \{(x_0, x_1, x_2, \ldots) \mid \forall n \in \mathbb{N}.\, x_n \in X_n \wedge f_n(x_{n+1}) = x_n\}.$$

(ii) *Each polynomial functor* $F\colon \mathbf{Sets} \to \mathbf{Sets}$ *that is constructed without powerset is* ω*-continuous.*

Proof. (i) The maps $\zeta_n\colon Z \to X_n$ are the n-th projections. The universal property is easily established: given a set Y with functions $g_n\colon Y \to X_n$ satisfying $f_{n+1} \circ g_{n+1} = g_n$, the unique map $h\colon Y \to Z$ with $\zeta_n \circ h = g_n$ is given by the ω-tuple $h(y) = (g_0(y), g_1(y), g_2(y), \ldots)$.

(ii) By induction on the structure of F, using that products, coproducts and (constant) exponents preserve the relevant constructions. ❑

4.4.3. Corollary. *Each polynomial functor* $F\colon \mathbf{Sets} \to \mathbf{Sets}$ *without powerset has a final coalgebra, which can be computed as in (4.5).* ❑

We conclude this subsection on ω-continuous functors by actually (re)calculating a final coalgebra.

4.4.4. Example. In Corolary 2.3.6 (ii) we have seen that the final coalgebra for a (simple polynomial) functor $F(X) = X^A \times 2$ can be described as the set $2^{A^\star} = \mathcal{P}(A^\star) = \mathcal{L}(A)$ of languages with alphabet A. Here we shall reconstruct this coalgebra as limit of an ω-chain.

Therefore we start by investigating what the chain (4.5) looks like for this functor F.

$$\begin{array}{rcl} F^0(1) & = & 1 \\ F^1(1) & = & 1^A \times 2 \;\cong\; 1 \times 2 \;\cong\; 2 \\ F^2(1) & \cong & 2^A \times 2 \;\cong\; 2^{A+1} \\ F^3(1) & \cong & \big(2^{A+1}\big)^A \times 2 \;\cong\; 2^{A\times(A+1)} \times 2 \;\cong\; 2^{A^2+A+1} \qquad \textit{etc.} \end{array}$$

One sees that:

$$F^n(1) \;\cong\; \mathcal{P}\big(\textstyle\sum_{i=0}^{n-1} A^i\big).$$

And also that the maps $F^n(!)\colon F^{n+1}(1) \to F^n(1)$ are given by the inverse image κ_n^{-1} of the obvious coprojection function $\kappa_n\colon A^{n-1} + \cdots + 1 \longrightarrow A^n + A^{n-1} + \cdots + 1$. An element $U \in Z$ of the limit Z as described in Lemma 4.4.2 (i) consists of elements $U_n \subseteq A^{n-1} + \cdots + 1$, with the requirement that $\kappa_n^{-1}(U_{n+1}) = U_n$. The latter means that these $U_{n+1} \subseteq A^n + A^{n-1} + \cdots + 1$ can be identified with subsets $U_{n+1} \subseteq A^n$ of words of length n. Together they form a set of words, or language, $U \subseteq A^\star$, like in the original description in Corolary 2.3.6 (ii).

4.4.2 Final coalgebras for ω-accessible functors

There is a general notion of "accessible" category and "accessible" functor, see *e.g.* [45]. But we only need the special case of "ω-accessibility" on $\mathbf{Sets}$ for our main result: every finite polynomial functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$ has a final coalgebra $Z \xrightarrow{\cong} F(Z)$. This special case uses that every set is a (directed) union of its finite subsets.

4.4.5. Definition. A functor $F\colon \mathbf{Sets} \to \mathbf{Sets}$ is ω-accessible (or finitary) if it satisfies for each set X,

$$F(X) \;=\; \bigcup_{U\in\mathcal{P}_{\mathrm{fin}}(X)} F(U) \;=\; \bigcup\{F(U) \mid U \subseteq X \text{ finite}\}.$$

This means that F is determined by how it acts on finite sets.

4.4.6. Lemma. *Each finite polynomial functor* $F\colon \mathbf{Sets} \to \mathbf{Sets}$ *is* ω*-accessible.*

Proof. By induction on the structure of F, using that F preserves inclusions. This accounts for the inclusion $(\supseteq)$. For the reverse inclusion we shall consider two cases, namely lists and finite powersets.

- If $F = G^\star$ and $\langle y_0, \ldots, y_n\rangle \in F(X) = G(X)^\star$, then by induction assumption there are finite subsets $U_i \subseteq X$ with $y_i \in G(U_i)$. The union $U = \bigcup_i U_i$ is then a finite set with $y_i \in G(U)$ for each i. This yields $\langle y_0, \ldots, y_n\rangle \in G(U)^\star = F(U)$, as required.
- The same argument applies for the case $F = \mathcal{P}_{\mathrm{fin}}(G)$ and an element $\{y_0, \ldots, y_n\} \in F(X) = \mathcal{P}_{\mathrm{fin}}(G(X))$. ❑

The temporal logic of coalgebras from the previous section is used in the following observation.

4.4.7. Lemma. *For each coalgebra* $c\colon X \to F(X)$ *of a finite polynomial functor* F*, and for each of its states* $x \in X$*, the invariant predicate of successor states of* x*,*

$$\underset{\leftarrow}{\Diamond}(x) \;=\; \{y \in X \mid x \longrightarrow^* y\}$$

is countable.

Proof. By Exercise 4.3.3 we have $\underset{\leftarrow}{\Diamond}(x) = \bigcup_{n\in\mathbb{N}} \underset{\leftarrow}{\bigcirc}^n(\cdot = x)$. Hence it suffices to show that for each $x \in X$ the set $\underset{\leftarrow}{\bigcirc} x = \{y \mid x \longrightarrow y\} = \underset{\leftarrow}{\mathrm{Pred}}(F)(\coprod_c(\cdot = x)) = \underset{\leftarrow}{\mathrm{Pred}}(F)(\{c(x)\})$ of immediate successors is finite. Now we use that F is ω-accessible and that predicate lifting is basically given by functor application (see Lemma 4.1.4) in:

$$\begin{array}{rcl} c(x) \in F(X) = \displaystyle\bigcup_{U\in\mathcal{P}_{\mathrm{fin}}(X)} F(U) = \bigcup_{U\in\mathcal{P}_{\mathrm{fin}}(X)} \mathrm{Pred}(F)(U) & & \\ \Longrightarrow & \{c(x)\} \subseteq \mathrm{Pred}(F)(U), & \text{for some finite } U \subseteq X \\ \Longrightarrow & \underset{\leftarrow}{\mathrm{Pred}}(F)(\{c(x)\}) \subseteq U, & \text{for some finite } U \subseteq X, \text{ since } \underset{\leftarrow}{\mathrm{Pred}}(F) \dashv \mathrm{Pred}(F) \\ \Longrightarrow & \underset{\leftarrow}{\mathrm{Pred}}(F)(\{c(x)\}) \text{ is finite.} & \end{array}$$ ❑

We can now prove the main result, based on [29], see also [216].

4.4.8. Theorem (Theorem 2.3.9). *Each finite Kripke polynomial functor* $F\colon \mathbf{Sets} \to \mathbf{Sets}$ *has a final coalgebra.*

Proof. The proof proceeds in two steps: first a so-called weakly final coalgebra is constructed, and then a properly final one is obtained via a suitable quotient.

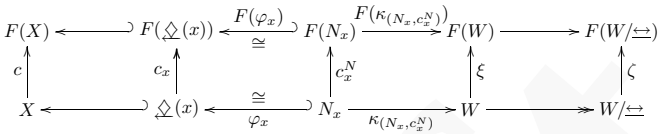
We first form the collection of all states and F-coalgebras on subsets of $\mathbb{N}$.

$$W \;=\; \{(n, N, d\colon N \to F(N)) \mid n \in N \wedge N \subseteq \mathbb{N}\}.$$

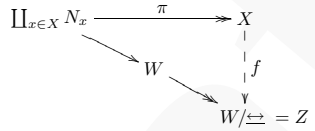
It carries a coalgebra structure $\xi\colon W \to F(W)$ itself, such that for each coalgebra $d\colon N \to F(N)$ on $N \subseteq \mathbb{N}$ the associated coprojection $\kappa_{(N,d)}\colon N \to W$ given by $n \mapsto (n, N, d)$ is a homomorphism. Just take $\xi(n, N, d) = (F(\kappa_{N,d}) \circ d)(n)$. Next we take $Z = W/\!\leftrightarrows$. Quotienting ξ with bisimilarity $\leftrightarrows$, like in Exercise 3.4.1, forces the induced coalgebra $\zeta\colon Z = W/\!\leftrightarrows \to F(Z)$ to be observable. The latter means that bisimilarity on Z is equality. We claim that ζ is final.

For each element $x \in X$, we know by the previous lemma that the set $\underset{\leftarrow}{\Diamond}(x) = \{y \in X \mid x \longrightarrow^* y\}$ of successor states is countable. Therefore it is isomorphic to a subset of $N_x \subseteq \mathbb{N}$, say via $\varphi_x\colon N_x \xrightarrow{\cong} \underset{\leftarrow}{\Diamond}(x)$. The invariant $\underset{\leftarrow}{\Diamond}(x) \subseteq X$ carries a subcoalgebra structure, say $c_x\colon \underset{\leftarrow}{\Diamond}(x) \to F(\underset{\leftarrow}{\Diamond}(x))$, by Theorem 4.2.5 (i). We write $c_x^N\colon N_x \to F(N_x)$

for the induced coalgebra structure on N_x. This leads to the following diagram of coalgebra maps.



Having chosen for each $x \in X$ such a subcoalgebra $c^N_X \colon N_x \to F(N_x)$ of c we can form their coproduct coalgebra. Its state space $\coprod_{x\in X} N_x$ is the coproduct in $\mathbf{Sets}$, see Exercise 2.1.13, with cotuple homomorphism $\pi \colon \coprod_{x\in X} N_x \twoheadrightarrow X$. This gives the following diagram of coalgebra homomorphisms.



The dashed homomorphism f exists because the epimorphism π is the coequaliser of its kernel relation $\mathrm{Ker}(\pi)$, see Exercise 3.3.6. The latter is a bisimulation, by Proposition 3.2.6 (ii), so that the composite $\coprod_{x\in X} N_x \to W \twoheadrightarrow W/\!\leftrightarrows$ maps pairs $(u, v) \in \mathrm{Ker}(\pi)$ to equal values.

The homomorphism f is unique because the coalgebra on $Z = W/\!\leftrightarrows$ is observable by construction (see Exercises 3.4.1 and 3.4.2): if there are two homomorphisms $f, g \colon X \to Z$, their image $\mathrm{Im}(\langle f, g\rangle) \subseteq Z \times Z$ is a bisimulation, and is thus contained in the equality relation on Z. Hence $(f(x), g(x)) \in \mathrm{Im}(\langle f, g\rangle) \subseteq \mathrm{Eq}(Z)$, so that $f(x) = g(x)$, for all $x \in X$. ❑

More information about this construction for ω-accessible functors may be obtained from [242, 157, 9].

Weakly final (co)algebras may also be constructed in (second order) polymorphic type theory, see [106, 243]. Under suitable parametricity conditions, these constructions yield proper final coalgebras, see [108, 197, 25].

Exercises

4.4.1. Fill in the details of the proof of Lemma 4.4.1.

4.4.2. (i) Formulate the notions of ω-colimit and ω-cocontinuity.

(ii) Check that the colimit of an ω-chain $X_0 \xrightarrow{f_0} X_1 \xrightarrow{f_1} X_2 \cdots$ in $\mathbf{Sets}$ can be described as quotient of the disjoint union:

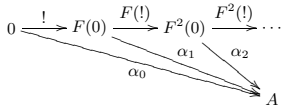
$$\coprod_{n\in\mathbb{N}} X_n/\!\sim \quad = \quad \{(n, x) \mid n \in \mathbb{N} \wedge x \in X_n\}/\!\sim$$

where

$$(n, x) \sim (m, y) \iff \exists p \geq n, m.\ f_{np}(x) = f_{mp}(y),$$

with $f_{qp} = f_{p-1} \circ f_{p-2} \circ \cdots \circ f_q \colon X_q \to X_p$ for $q \leq p$.

(iii) Prove that, dually to Lemma 4.4.1, the initial algebra of an ω-cocontinuous functor $F \colon \mathbb{C} \to \mathbb{C}$ on a category $\mathbb{C}$ with initial object $0 \in \mathbb{C}$ can be obtained as ω-colimit A of the chain:



with the induced initial algebra $\alpha \colon F(A) \xrightarrow{\cong} A$ satisfying $\alpha \circ F(\alpha_n) = \alpha_{n+1}$.

4.4.3. Consider an initial algebra $\alpha \colon F(A) \xrightarrow{\cong} A$ as constructed in the previous exercise for a functor F that preserves monomorphisms (like any weak-pullback-preserving, and hence any polynomial functor, see Exercise 3.3.7). Assume F also has a final coalgebra $\zeta \colon Z \xrightarrow{\cong} F(Z)$, and let $\iota \colon A \to Z$ be the unique (algebra and coalgebra) homomorphism with $\zeta \circ \iota = F(\iota) \circ \alpha^{-1}$. Prove that ι is injective.

[*Hint.* Define suitable $\zeta_n \colon Z \to F^n(1)$ and use that $F^n(!) \colon F^n(0) \to F^n(1)$ is mono.]

4.5 Trace semantics

This section will describe how finite traces of observables of computations can be described via coinduction. The two main examples involve non-deterministic automata and context-free grammars. This requires a move from the category $\mathbf{Sets}$ of sets and functions to the category **REL** of sets and relations, as introduced in Example 1.4.2 (iv). Its objects are sets, and its morphisms $X \to Y$ are relations $R \subseteq X \times Y$. They may equivalently be described as a characteristic function $R \colon X \to \mathcal{P}(Y)$. We shall often switch back-and-forth between these two notations. The identity map $X \to X$ in **REL** is the equality relation $\mathrm{Eq}(X) \subseteq X \times X$. And composition of $R \colon X \to Y$ and $S \colon Y \to Z$ is the usual relational composition: $S \circ R = \{(x, z) \in X \times Z \mid \exists y \in Y.\, R(x, y) \wedge S(y, z)\}$. There is an obvious "graph" functor $\mathbf{Sets} \to$ **REL** as already described in Example 1.4.4 (iv). It maps a set to itself, and a function $f \colon X \to Y$ to its graph relation $\mathrm{Graph}(f) = \{(x, y) \in X \times Y \mid f(x) = y\}$.

We start with an example. Let us fix a set A, and write L for the "list" functor $\mathbf{Sets} \to \mathbf{Sets}$ given by $X \mapsto 1 + (A \times X)$. We shall consider transition systems of the form $c \colon X \to \mathcal{P}(LX) = \mathcal{P}(1 + (A \times X))$. As noticed in Exercise 2.2.3, such transition systems correspond to non-deterministic automata $X \to \mathcal{P}(X)^A \times 2$. Here we prefer the formulation with the powerset on the outside, because it allows us to describe such automata as maps $X \to L(X)$ in the category **REL**.

These transition systems $X \to \mathcal{P}(1 + (A \times X))$ describe besides usual transitions $x \xrightarrow{a} x'$, as shorthand for $(a, x') \in c(x)$, also terminating transitions $x \longrightarrow *$ for $* \in c(x)$. A **trace** for such a coalgebra starting at an element $x \in X$ consists of a list of elements $\langle a_1, \ldots, a_n\rangle \in A^\star$ for which there is a list of states $x_0, \ldots, x_n \in X$ where:

- $x_0 = x$;
- $x_i \xrightarrow{a_i} x_{i+1}$ for all $i < n$;
- $x_n \longrightarrow *$.

One can then define a function $\mathsf{trace}_c \colon X \to \mathcal{P}(A^\star)$ that maps a state to the set of traces starting in that state. We shall show how to define this function by coinduction. It turns out not to be a coincidence that the set $A^\star$ of lists is the initial algebra of the functor L.

First we need to move from sets to relations as morphisms. We shall lift the functor $L \colon \mathbf{Sets} \to \mathbf{Sets}$ to $L \colon$ **REL** $\to$ **REL**. It maps an object (or set) X to $L(X)$, and a morphism $R \colon X \to Y$ in **REL** to the morphism $\mathrm{Rel}(L)(R) \colon L(X) \to L(Y)$ obtained by relation lifting. This yields a functor because relation lifting preserves equality and composition, see Lemma 3.2.1. For convenience, we note that according to Definition 3.1.1, the relation lifting $\mathrm{Rel}(L)(R) \subseteq L(X) \times L(Y)$ is described by:

$$\mathrm{Rel}(L)(R) \quad = \quad \{(*, *)\} \cup \{\langle (a, x), (a, x')\rangle \mid R(x, x')\}.$$

We now have three important observations.

1. The first one is simple: a transition system, or $\mathcal{P}(L)$-coalgebra, $c \colon X \to \mathcal{P}(LX)$ as considered above is an L-coalgebra $X \to LX$ in **REL**, for the lifted functor $L \colon$ **REL** $\to$ **REL**.

2. Further, the above trace map $\mathsf{trace}_c \colon X \to \mathcal{P}(A^\star)$, considered as a homomorphism $X \to A^\star$ in **REL**, is a homomorphism of L-coalgebras in **REL**:

$$\begin{array}{ccc} L(X) & \xrightarrow{\mathrm{Rel}(L)(\mathsf{trace}_c)} & L(A^\star) \\ {\scriptstyle c}\uparrow & & \cong \uparrow{\scriptstyle \mathrm{Graph}(\alpha^{-1})} \\ X & \xrightarrow[\mathsf{trace}_c]{} & A^\star \end{array} \tag{4.6}$$

where $\alpha = [\mathsf{nil}, \mathsf{cons}] \colon L(A^\star) \xrightarrow{\cong} A^\star$ is the initial algebra map. Commutation amounts to:

$$\sigma \in \mathsf{trace}_c(x) \iff \exists u \in L(X).\, u \in c(x) \wedge \langle u, \alpha^{-1}(\sigma)\rangle \in \mathrm{Rel}(L)(\mathsf{trace}_c).$$

This may be split in two cases, depending on whether the list $\sigma \in A^\star$ is empty or not:

$$\begin{array}{rcl} \langle\rangle \in \mathsf{trace}_c(x) & \iff & \exists u \in L(X).\, u \in c(x) \wedge \langle u, * \rangle \in \mathrm{Rel}(L)(\mathsf{trace}_c) \\ & \iff & * \in c(x) \\ & \iff & x \longrightarrow * \\ a \cdot \sigma \in \mathsf{trace}_c(x) & \iff & \exists u \in L(X).\, u \in c(x) \wedge \langle u, (a,\sigma)\rangle \in \mathrm{Rel}(L)(\mathsf{trace}_c) \\ & \iff & \exists x' \in X.\, (a, x') \in c(x) \wedge \langle x', \sigma\rangle \in \mathsf{trace}_c \\ & \iff & \exists x' \in X.\, x \xrightarrow{a} x' \wedge \sigma \in \mathsf{trace}_c(x'). \end{array}$$

This shows that the map trace_c indeed makes the diagram commute.

3. Finally, the map trace_c in (4.6) is obtained by coinduction since the graph of the initial L-algebra $\mathrm{Graph}(\alpha^{-1}) \colon A^\star \xrightarrow{\cong} L(A^\star)$ is a final L-coalgebra in **REL**.

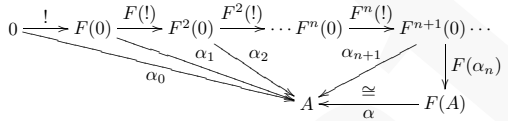
We have already seen that there is a homomorphism trace_c for an arbitrary L-coalgebra c. Here we check that it is unique: if a relation $S \subseteq X \times A^\star$ also satisfies $\mathrm{Graph}(\alpha^{-1}) \circ S = \mathrm{Rel}(L)(S) \circ c$, then $S = \mathsf{trace}_c$. Indeed, for $x \in X$ one obtains $\sigma \in S(x) \Leftrightarrow \sigma \in \mathsf{trace}_c(x)$ by induction on $\sigma \in A^\star$:

$$\begin{array}{rcl} \langle\rangle \in S(x) & \iff & \exists \tau \in A^\star.\, * = \alpha^{-1}(\tau) \wedge \tau \in S(x) \\ & \iff & \exists \tau \in A^\star.\, * \in \mathrm{Graph}(\alpha^{-1})(\tau) \wedge \tau \in S(x) \\ & \iff & * \in (\mathrm{Graph}(\alpha^{-1}) \circ S)(x) \\ & \iff & * \in (\mathrm{Rel}(L)(S) \circ c)(x) \\ & \iff & \exists u \in L(X).\, * \in \mathrm{Rel}(L)(S)(u) \wedge u \in c(x) \\ & \iff & * \in c(x) \\ & \iff & \langle\rangle \in \mathsf{trace}_c(x) \\ a \cdot \sigma \in S(x) & \iff & \exists \tau \in A^\star.\, (a,\sigma) = \alpha^{-1}(\tau) \wedge \tau \in S(x) \\ & \iff & \exists \tau \in A^\star.\, (a,\sigma) \in \mathrm{Graph}(\alpha^{-1})(\tau) \wedge \tau \in S(x) \\ & \iff & (a,\sigma) \in (\mathrm{Graph}(\alpha^{-1}) \circ S)(x) \\ & \iff & (a,\sigma) \in (\mathrm{Rel}(L)(S) \circ c)(x) \\ & \iff & \exists u \in L(X).\, (a,\sigma) \in \mathrm{Rel}(L)(S)(u) \wedge u \in c(x) \\ & \iff & \exists x' \in X.\, \sigma \in S(x') \wedge (a, x') \in c(x) \\ & \overset{\text{(IH)}}{\iff} & \exists x' \in X.\, \sigma \in \mathsf{trace}_c(x') \wedge x \xrightarrow{a} x' \\ & \iff & a \cdot \sigma \in \mathsf{trace}_c(x). \end{array}$$

We shall describe this situation in greater generality, following [109]. Like before, for a polynomial functor $F \colon \mathbf{Sets} \to \mathbf{Sets}$ we shall also write F for the lifting $F \colon \mathbf{REL} \to \mathbf{REL}$ that maps a set X to FX and a relation $R \colon X \to Y$ to its relation lifting $\mathrm{Rel}(F)(R) \colon FX \to FY$.

4.5.1. Theorem. *Let $F \colon \mathbf{Sets} \to \mathbf{Sets}$ be a polynomial functor without powersets or exponents $(-)^B$ with infinite sets B. The initial F-algebra $\alpha \colon F(A) \xrightarrow{\cong} A$ in $\mathbf{Sets}$ then yields a final F-coalgebra $\mathrm{Graph}(\alpha^{-1}) \colon A \xrightarrow{\cong} F(A)$ in **REL**, for the lifting $F \colon$ **REL** $\to$ **REL**.*

Proof. For an F-coalgebra $c \colon X \to FX$ in **REL**—or a function $c \colon X \to \mathcal{P}(FX)$—we shall construct an appropriate map $\mathsf{trace}_c \colon X \to \mathcal{P}(A)$. We use that the initial algebra $\alpha \colon F(A) \xrightarrow{\cong} A$ is obtained, like in Exercise 4.4.2, as colimit of the ω-chain:



where the coprojections $\alpha_n \colon F^n(0) \to A$ satisfy $\alpha \circ F(\alpha_n) = \alpha_{n+1}$ by construction of α.

In this situation we first define a collection of relations $S_n \subseteq X \times F^n(0)$ by induction:

$$S_0 \;=\; 0 \qquad \text{and} \qquad S_{n+1} \;=\; \mathrm{Rel}(F)(S_n) \circ c$$

Hence S_0 is the empty relation $\langle !_X, \mathrm{id}\rangle \colon 0 \rightarrowtail X \times 0$, which can also be described as inverse image of the equality relation on X, namely $S_0 = (\mathrm{id} \times !_X)^{-1}(\mathrm{Eq}(X))$, or as graph $\mathrm{Graph}(!_X)$ of the unique function $!_X \colon 0 \to X$. And S_{n+1} is the composite of $c \subseteq X \times F(X)$ and $\mathrm{Rel}(F)(S_n) \subseteq F(X) \times F^{n+1}(0)$. We claim that S_n is then a composition of n relations, of the form:

$$\begin{array}{rcl} S_n & = & \mathrm{Graph}(F^n(!_X)) \circ \mathrm{Rel}(F)^{n-1}(c) \circ \cdots \circ \mathrm{Rel}(F)(c) \circ c \\ & = & \{(x,a) \in X \times F^n(0) \mid \exists x_0 \in X, x_1 \in F(X), \ldots, x_n \in F^n(X). \\ & & \qquad x_0 = x \wedge \forall i < n.\, (x_i, x_{i+1}) \in \mathrm{Rel}(F)^n(c) \wedge x_n = F^n(!_X)(a)\}. \end{array} \tag{4.7}$$

This is obtained by induction on n, using that relation lifting preserves compositions and graphs.

We now define the relation $\mathsf{trace}_c \subseteq X \times A$ as follows.

$$\begin{array}{rcl} \mathsf{trace}_c & = & \bigcup_{n \in \mathbb{N}} \coprod_{\mathrm{id} \times \alpha_n} S_n \\ & = & \bigcup_{n \in \mathbb{N}} (\mathrm{Graph}(\alpha_n) \circ S_n). \end{array}$$

This is a union of an ascending chain: assume $(x,a) \in \coprod_{\mathrm{id}\times\alpha_n} S_n$, say $a = \alpha_n(b)$ for $b \in F^n(0)$, with $(x,b) \in S_n$. The latter yields $x_0 \in X$, $x_1 \in F(X)$, …, $x_n \in F^n(X)$ with $x_0 = x$, $\forall i < n.\, (x_i, x_{i+1}) \in \mathrm{Rel}(F)^i(c)$, and $x_n = F^n(!_{F0})(b)$. We take $b' = F^n(!_{F0})(b) \in F^{n+1}(0)$ and $x_{n+1} = F^{n+1}(!_X)(b') = F^n(!_{FX})(b) \in F^{n+1}(X)$. For (x,b') to be in S_{n+1} it suffices to show $(x_n, x_{n+1}) \in \mathrm{Rel}(F)^{n+1}(c)$. This is done as follows. We have $\mathrm{Eq}(0) \subseteq (!_X \times !_{FX})^{-1}(c)$, as demonstrated by the commuting diagram below.

$$\begin{array}{ccc} 0 & \xrightarrow{\;!\;} & c \\ {\scriptstyle \langle \mathrm{id}, \mathrm{id}\rangle}\downarrow & & \downarrow \\ 0 \times 0 & \xrightarrow[!_X \times !_{FX}]{} & X \times FX \end{array}$$

Then:

$$\begin{aligned}(b,b) \in \mathrm{Eq}(F^n(0)) &= \mathrm{Rel}(F)^n(\mathrm{Eq}(0)) \\ &\subseteq \mathrm{Rel}(F)^n((!_X \times !_{FX})^{-1}(c)) \\ &= (F^n(!_X) \times F^n(!_{FX}))^{-1}\mathrm{Rel}(F)^n(c).\end{aligned}$$

Hence the pair $x_n = F^n(!_X)(b)$, $x_{n+1} = F^n(!_{FX})(b)$ is in $\mathrm{Rel}(F)^n(c)$, as required. Finally, we have $\alpha_{n+1}(b') = \alpha_n(b) = a$, so that $(x,a) \in \coprod_{\mathrm{id}\times\alpha_{n+1}} S_{n+1}$. This shows that the chain is ascending.

Our aim is to show that trace_c is the unique map (in **REL**) making the following diagram commute.

$$\begin{array}{ccc} F(X) & \xrightarrow{\mathrm{Rel}(F)(\mathsf{trace}_c)} & F(A) \\ {\scriptstyle c}\uparrow & & \cong\uparrow{\scriptstyle \mathrm{Graph}(\alpha^{-1})} \\ X & \xrightarrow[\mathsf{trace}_c]{} & A \end{array}$$

Commutation of the above square is relatively easy, using that relation lifting for a functor F without powerset or infinite exponents preserves unions of ascending chains, see Exercise 3.2.1 (ii):

$$\begin{aligned}&\mathrm{Rel}(F)(\mathsf{trace}_c) \circ c \\ &= \big(\bigcup_{n\in\mathbb{N}} \coprod_{\mathrm{id}\times F(\alpha_n)} \mathrm{Rel}(F)(S_n)\big) \circ c \\ &= \bigcup_{n\in\mathbb{N}} \coprod_{\mathrm{id}\times F(\alpha_n)} \big(\mathrm{Rel}(F)(S_n) \circ c\big) && \text{by Exercise 4.5.2 (i)} \\ &= \bigcup_{n\in\mathbb{N}} \coprod_{\mathrm{id}\times F(\alpha_n)} S_{n+1} \\ &= \bigcup_{n\in\mathbb{N}} \coprod_{\mathrm{id}\times\alpha^{-1}} \coprod_{\mathrm{id}\times\alpha_{n+1}} S_{n+1} && \text{since } \alpha \circ F(\alpha_n) = \alpha_{n+1} \\ &= \coprod_{\mathrm{id}\times\alpha^{-1}} \bigcup_{n\in\mathbb{N}} \coprod_{\mathrm{id}\times\alpha_{n+1}} S_{n+1} && \text{since } \coprod_{\mathrm{id}\times\alpha^{-1}} \text{ is a left adjoint} \\ &= \coprod_{\mathrm{id}\times\alpha^{-1}} \big(\bigcup_{n\in\mathbb{N}} \coprod_{\mathrm{id}\times\alpha_{n+1}} S_{n+1} \cup \coprod_{\mathrm{id}\times\alpha_0} S_0\big) && \text{since } S_0 = 0 \\ &= \coprod_{\mathrm{id}\times\alpha^{-1}} \mathsf{trace}_c \\ &= \mathrm{Graph}(\alpha^{-1}) \circ \mathsf{trace}_c && \text{by Exercise 4.5.2 (iii).}\end{aligned}$$

The next step is uniqueness: assume $R \subseteq X \times A$ also satisfies $\mathrm{Rel}(F)(R) \circ c = \mathrm{Graph}(\alpha^{-1}) \circ R$, or equivalently, $R = \mathrm{Graph}(\alpha) \circ \mathrm{Rel}(F)(R) \circ c$. Then $R = \mathsf{trace}_c$, which is shown in two steps.

($\supseteq$) This requires inclusions $\mathrm{Graph}(\alpha_n) \circ S_n \subseteq R$ for each $n \in \mathbb{N}$. The proof method is induction, of course. The base case $n = 0$ is trivial: $\mathrm{Graph}(\alpha_0) \circ S_0 = \mathrm{Graph}(\alpha_0) \circ 0 = 0 \subseteq R$. For the induction step we compute:

$$\begin{aligned}\mathrm{Graph}(\alpha_{n+1}) \circ S_{n+1} &= \mathrm{Graph}(\alpha \circ F(\alpha_n)) \circ \mathrm{Rel}(F)(S_n) \circ c \\ &= \mathrm{Graph}(\alpha) \circ \mathrm{Graph}(F(\alpha_n)) \circ \mathrm{Rel}(F)(S_n) \circ c \\ &= \mathrm{Graph}(\alpha) \circ \mathrm{Rel}(F)(\mathrm{Graph}(\alpha_n)) \circ \mathrm{Rel}(F)(S_n) \circ c \\ &= \mathrm{Graph}(\alpha) \circ \mathrm{Rel}(F)(\mathrm{Graph}(\alpha_n) \circ S_n) \circ c \\ &\overset{\text{(IH)}}{\subseteq} \mathrm{Graph}(\alpha) \circ \mathrm{Rel}(F)(R) \circ c \\ &= R.\end{aligned}$$

($\subseteq$) For $(x,a) \in R$ we need to find an $n \in \mathbb{N}$ with $(x,a) \in \mathrm{Graph}(\alpha_n) \circ S_n$. Since $a \in A$, and A is obtained as colimit of the ω-chain $F^i(0)$, there is an $i \in \mathbb{N}$ and $b \in F^i(0)$ with $a = \alpha_i(b)$. So we are done if we can prove for each n,

$$\begin{aligned}\mathrm{Graph}(\alpha_n) \circ S_n &\supseteq R \cap (X \times \mathrm{Im}(\alpha_n)) \\ &= \mathrm{Graph}(\alpha_n) \circ \mathrm{Graph}(\alpha_n)^{-1} \circ R.\end{aligned} \tag{4.8}$$

We shall prove this inclusion by induction on $n \in \mathbb{N}$. The base case is trivial because $\mathrm{Im}(\alpha_0) = \emptyset$. For the induction step we reason as follows.

$$\begin{aligned}&\mathrm{Graph}(\alpha_{n+1}) \circ \mathrm{Graph}(\alpha_{n+1})^{-1} \circ R \\ &= \mathrm{Graph}(\alpha \circ F(\alpha_n)) \circ \mathrm{Graph}(\alpha \circ F(\alpha_n))^{-1} \circ R \\ &= \mathrm{Graph}(\alpha) \circ \mathrm{Rel}(F)(\mathrm{Graph}(\alpha_n)) \circ \mathrm{Rel}(F)(\mathrm{Graph}(\alpha_n))^{-1} \circ \mathrm{Graph}(\alpha)^{-1} \circ \\ &\qquad \mathrm{Graph}(\alpha) \circ \mathrm{Rel}(F)(R) \circ c \\ &= \mathrm{Graph}(\alpha) \circ \mathrm{Rel}(F)(\mathrm{Graph}(\alpha_n)) \circ \mathrm{Rel}(F)(\mathrm{Graph}(\alpha_n))^{-1} \circ \mathrm{Rel}(F)(R) \circ c \\ &= \mathrm{Graph}(\alpha) \circ \mathrm{Rel}(F)(\mathrm{Graph}(\alpha_n) \circ \mathrm{Graph}(\alpha_n)^{-1} \circ R) \circ c \\ &\overset{\text{(IH)}}{\subseteq} \mathrm{Graph}(\alpha) \circ \mathrm{Rel}(F)(\mathrm{Graph}(\alpha_n) \circ S_n) \circ c \\ &= \mathrm{Graph}(\alpha) \circ \mathrm{Graph}(F(\alpha_n)) \circ \mathrm{Rel}(F)(S_n) \circ c \\ &= \mathrm{Graph}(\alpha_{n+1}) \circ S_{n+1}.\end{aligned}$$

❑

This theorem describes traces for transition systems as described in the beginning of this section. The main application of [109] is the parsed language associated with a context free grammar.

4.5.2. Example. Recall from Subsection 2.2.5 that a context-free grammar (CFG) is described coalgebraically as a function $g\colon X \to \mathcal{P}((X+A)^\star)$ where X is the state space of nonterminals, and A is the alphabet of terminals (or tokens). Such a CFG is thus a coalgebra in **REL** of the functor $F(X) = (X+A)^\star$. Let us write its initial algebra as:

$$(A^\triangle + A)^\star \xrightarrow[\cong]{\alpha} A^\triangle$$

Notice that $F(X) = (X+A)^\star = \coprod_{n\in\mathbb{N}} (X+A)^n = \coprod_{\sigma\in(1+A)^\star} X^{\|\sigma\|}$ where $\|\sigma\| \in \mathbb{N}$ is the number of $* \in 1$ occurring in $\sigma \in (1+A)^\star$. Hence an F-algebra $F(X) \to X$ involves an n-ary operation $X^n \to X$ for each $\sigma \in (1+A)^\star$ with $n = \|\sigma\|$. An example of such a σ is $\langle \mathsf{if}, *, \mathsf{then}, *, \mathsf{else}, *, \mathsf{fi}\rangle$. It may be understood as a ternary operation. The initial algebra $A^\triangle$ then consists of terms built with such operations. Hence it contains all structured or parsed words over A.

Theorem 4.5.1 gives for each CFG $g\colon X \to \mathcal{P}((X+A)^\star)$ a unique homomorphism $\mathsf{trace}_g\colon X \to \mathcal{P}(A^\triangle)$. It maps a nonterminal $x \in X$ to the collection of parsed words that can be produced from x.

The trace semantics that we have described in this section induces a new type of equivalence on states (of suitable coalgebras), namely trace equivalence, given by the relation $\{(x,y) \mid \mathsf{trace}(x) = \mathsf{trace}(y)\}$. Exercise 4.5.5 below shows that bisimilarity induces trace equivalence, but not the other way around.

The category **REL** of sets and relations that we use for trace semantics may be described as the so-called Kleisli category of the powerset monad. This aspect of trace semantics is also present in other sources [203, 137] on this topic, and may form the basis for a more systematic investigation.

Exercises

4.5.1. Give an explicit description of the relations S_n from (4.7) for the functor $L = 1 + (A \times (-))$ from the beginning of this section.

4.5.2. Prove that for relations R_i, R, S and functions f:
(i) $\big(\bigcup_{i\in\mathbb{N}} R_i\big) \circ S = \bigcup_{i\in\mathbb{N}} \big(R_i \circ S\big)$.

(ii) $\left(\coprod_{\mathrm{id}\times f} R\right) \circ S = \coprod_{\mathrm{id}\times f} \left(R \circ S\right)$.

(iii) $\mathrm{Graph}(f) \circ R = \coprod_{\mathrm{id}\times f} R$.

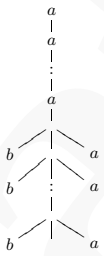
4.5.3. Consider the alphabet $\{a, b\}$ with two non-terminals V, W and productions:

$$V \longrightarrow a \cdot V \qquad V \longrightarrow W \qquad W \longrightarrow b \cdot W \cdot a \qquad W \longrightarrow \langle\rangle.$$

(i) Describe this CFG as a coalgebra with state space $\{V, W\}$.

(ii) Check that the unparsed language generated by V is the set $\{a^n b^m a^m \mid n, m \in \mathbb{N}\}$.

(iii) Consider the parsed language map $\mathsf{trace}_g \colon \{V, W\} \to \{a, b\}^{\triangle}$ from Example 4.5.2 and show that a term $t \in \mathsf{trace}_g(V)$ can be drawn as a tree:



4.5.4. (i) Define a function $A^{\triangle} \to A^{\star}$ that maps parsed words to "flat" words by initiality. Prove that this function is a split epi.

(ii) Prove that the assignment $A \mapsto A^{\triangle}$ is functorial, and that the mapping $A^{\triangle} \twoheadrightarrow A^{\star}$ from (i) form a natural transformation.

(iii) Let $A^{\wedge}$ be the final coalgebra of the functor $X \mapsto (X + A)^{\star}$ from Example 4.5.2. It consists of both the finite and infinite parsed words. Define a map $A^{\triangle} \to A^{\wedge}$ and show that it is injective.

[This gives the following fundamental span of languages $A^{\star} \twoheadleftarrow A^{\triangle} \rightarrowtail A^{\wedge}$. The three operations involved $A \mapsto A^{\star}, A^{\triangle}, A^{\wedge}$ are all "monads" (see Subsection 5.2.1) and the mappings between them preserve the monad structure, see [109].]

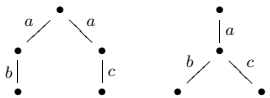
4.5.5. Assume two coalgebras $c \colon X \to \mathcal{P}(FX)$ and $d \colon Y \to \mathcal{P}(FY)$, where F is a functor with powersets and infinite exponents (like in Theorem 4.5.1).

(i) Assume $f \colon X \to Y$ is a homomorphisms (of $\mathcal{P}F$-coalgebras). Prove that $\mathsf{trace}_d \circ f = \mathsf{trace}_c$.

(ii) Conclude that bisimilarity is included in trace equivalence:

$$x \underset{c}{\leftrightarrow}_d y \quad \Longrightarrow \quad \mathsf{trace}_c(x) = \mathsf{trace}_d(y).$$

(iii) Check that the reverse implication ($\Longleftarrow$) in (ii) does not hold, for instance via the following two pictures.



Bibliography

[1] M. Abott, Th. Altenkirch, N. Ghani, and C. McBride. Categories of containers. In A.D. Gordon, editor, *Foundations of Software Science and Computation Structures*, number 2620 in Lect. Notes Comp. Sci., pages 23–38. Springer, Berlin, 2003.

[2] M. Abott, Th. Altenkirch, N. Ghani, and C. McBride. Derivatives of containers. In M. Hofmann, editor, *Typed Lambda Calculi and Applications*, number 2701 in Lect. Notes Comp. Sci., pages ??–?? Springer, Berlin, 2003.

[3] S. Abramsky. A domain equation for bisimulation. *Inf. & Comp.*, 92:161–218, 1990.

[4] P. Aczel. *Non-well-founded sets*. CSLI Lecture Notes 14, Stanford, 1988.

[5] P. Aczel. Final universes of processes. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics*, number 802 in Lect. Notes Comp. Sci., pages 1–28. Springer, Berlin, 1994.

[6] P. Aczel, J. Adámek, S. Milius, and J. Velebil. Infinite trees and completely iterative theories: a coalgebraic view. *Theor. Comp. Sci.*, 300 (1-3):1–45, 2003.

[7] 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.

[8] J. Adámek. Observability and nerode equivalence in concrete categories. In F. Gécseg, editor, *Fundamentals of Computation Theory*, number 117 in Lect. Notes Comp. Sci., pages 1–15. Springer, Berlin, 1981.

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

[10] J. Adámek and V. Koubek. On the greatest fixed point of a set functor. *Theor. Comp. Sci.*, 150:57–75, 1995.

[11] J. Adámek and S. Milius, editors. *Coalgebraic Methods in Computer Science (CMCS'04)*, number 106 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2004.

[12] L. Adleman. Computing with DNA. *Scientific American*, 279(2):54–61, 1998.

[13] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, Reading, Massachusetts, 1985.

[14] M.A. Arbib. *Theories of Abstract Automata*. Prentice Hall, 1969.

[15] M.A. Arbib and E.G. Manes. Foundations of system theory: Decomposable systems. *Automatica*, 10:285–302, 1974.

[16] M.A. Arbib and E.G. Manes. Adjoint machines, state-behaviour machines, and duality. *Journ. of Pure & Appl. Algebra*, 6:313–344, 1975.

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

[18] M.A. Arbib and E.G. Manes. Foundations of system theory: the Hankel matrix. *Journ. Comp. Syst. Sci*, 20:330–378, 1980.

[19] M.A. Arbib and E.G. Manes. Generalized Hankel matrices and system realization. *SIAM J. Math. Analysis*, 11:405–424, 1980.

[20] M.A. Arbib and E.G. Manes. Machines in a category. *Journ. of Pure & Appl. Algebra*, 19:9–20, 1980.

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

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

[23] K. Arnold and J. Gosling. *The Java Programming Language*. The Java Series. Addison-Wesley, 2nd edition, 1997.

[24] E.S. Bainbridge. *A unified minimal realization theory with duality*. PhD thesis, Univ. Michigan, Ann Arbor, 1972. Techn. rep. 140, Dep. of Comp. and Comm. Sci.

[25] E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott. Functorial polymorphism. *Theor. Comp. Sci.*, 70(1):35–64, 1990. Corrigendum in *Theor. Comp. Sci.* 71(3):431, 1990.

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

[27] L.S. Barbosa. Towards a calculus of state-based software components. *Journ. of Universal Comp. Sci.*, 9(8):891–909, 2003.

[28] H.P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North-Holland, Amsterdam, 2nd rev. edition, 1984.

[29] 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.

[30] M. Barr and Ch. Wells. *Toposes, Triples and Theories*. Springer, Berlin, 1985. Revised and corrected version available from URL: `www.cwru.edu/artsci/math/wells/pub/ttt.html`.

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

[32] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-Chr. Filliâtre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner. The Coq Proof Assistant User's Guide Version 6.1. Technical Report 203, INRIA Rocquencourt, France, May 1997.

[33] F. Bartels. *On generalised coinduction and probabilistic specification formats. Distributive laws in coalgebraic modelling*. PhD thesis, Free Univ. Amsterdam, 2004.

[34] F. Bartels, A. Sokolova, and E. de Vink. A hierarchy of probabilistic system types. *Theor. Comp. Sci.*, 327(1-2):3–22, 2004.

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

[36] J. van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic II*, pages 167–247, Dordrecht, 1984. Reidel.

[37] A. Benveniste, P. LeGuernic, and C. Jacquemot. Synchronous programming with events and relations: the SIGNAL language and its semantics. *Science of Comput. Progr.*, 16:103–149, 1991.

[38] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, Amsterdam, 2001.

[39] M. Bidoit and R. Hennicker. Proving the correctness of behavioural implementations. In V.S. Alagar and M. Nivat, editors, *Algebraic Methods and Software Technology*, number 936 in Lect. Notes Comp. Sci., pages 152–168. Springer, Berlin, 1995.

[40] M. Bidoit, R. Hennicker, and A. Kurz. On the duality between observability and reachability. In F. Honsell, editor, *Foundations of Software Science and Computation Structures*, Lect. Notes Comp. Sci. Springer, Berlin, 2001. To appear.

[41] R. Bird. *Introduction to Functional Programming using Haskell*. Prentice Hall Press, 2nd edition, 1998.

[42] R. Bird and O. de Moor. *Algebra of Programmming*. Prentice Hall Int. Series in Comput. Sci., 1996.

[43] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Number 53 in Tracts in Theor. Comp. Sci. Cambridge Univ. Press, 2001.

[44] S.L. Bloom and Z. Ésik. *Iteration Theories: The Equational Logic of Iterative Processes*. Number ?? in EATCS Monographs. Springer, Berlin, 1993.

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

[46] R. Brown. *Topology*. John Wiley & Sons, New York, 2nd rev. edition, 1988.

[47] K.B. Bruce, L. Cardelli, G. Castagna, The Hopkins Objects Group (J. Eifrig, S. Smith, V. Trifonov), G. Leavens, and B.C. Pierce. On binary methods. *Theory & Practice of Object Systems*, 1(3):221–242, 1996.

[48] J.A. Brzozowski. Derivatives of regular expressions. *Journ. ACM*, 11(4):481–494, 1964.

[49] P.J. Cameron. *Sets, Logic and Categories*. Undergraduate Mathematics. Springer, 1999.

[50] A. Carboni, G.M. Kelly, and R.J. Wood. A 2-categorical approach to change of base and geometric morphisms I. *Cah. de Top. et Géom. Diff.*, 32(1):47–95, 1991.

[51] C. Cïrstea. Coalgebra semantics for hidden algebra: parametrised objects and inheritance. In F. Parisi Presicce, editor, *Recent Trends in Data Type Specification*, number 1376 in Lect. Notes Comp. Sci., pages 174–189. Springer, Berlin, 1998.

[52] C. Cïrstea. Integrating observational and computational features in the specification of state-based dynamical systems. *Inf. Théor. et Appl.*, 35(1):1–29, 2001.

[53] J.R.B. Cockett and T. Fukushima. About Charity. Technical Report 92/480/18, Dep. Comp. Sci., Univ. Calgary, 1992.

[54] J.R.B. Cockett and D. Spencer. Strong categorical datatypes I. In R.A.G. Seely, editor, *Category Theory 1991*, number 13 in CMS Conference Proceedings, pages 141–169, 1992.

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

[56] R. Cockett. Deforestation, program transformation, and cut-elimination. In A. Corradini, M. Lenisa, and U. Montanari, editors, *Coalgebraic Methods in Computer Science*, number 44 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2001. `www.sciencedirect.com/science/journal/15710661`.

[57] A. Corradini, M. Lenisa, and U. Montanari, editors. *Coalgebraic Methods in Computer Science (CMCS'01)*, number 44 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2001.

[58] A. Corradini, M. Lenisa, and U. Montanari, editors. *Coalgebraic Methods in Computer Science*, volume 13(2) of *Math. Struct. in Comp. Sci.*, 2003. Special issue on CMCS'01.

[59] S. Coupet-Grimal and L. Jakubiec. Hardware verification using co-induction in COQ. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics*, number 1690 in Lect. Notes Comp. Sci., pages 91–108. Springer, Berlin, 1999.

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

[61] P. Cuoq and M. Pouzet. Modular causality in a synchronous stream language. In D. Sands, editor, *Programming Languages and Systems (ESOP)*, number 2028 in Lect. Notes Comp. Sci., pages 237–251. Springer, Berlin, 2001.

[62] N.J. Cutland. *Computability*. Cambridge Univ. Press, 1980.

[63] D. van Dalen, H.C. Doets, and H. de Swart. *Sets: Naive, Axiomatic and Applied*. Number 106 in Pure & applied Math. Pergamum Press, 1978.

[64] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Math. Textbooks. Cambridge Univ. Press, 1990.

[65] L.A. Dennis and A. Bundy. A comparison of two proof critics: Power vs. robustness. In V.A. Carreño, C.A. Muñoz, and S. Tahar, editors, *Theorem Proving in Higher Order Logics*, number 2410 in Lect. Notes Comp. Sci., pages 182–197. Springer, Berlin, 2002.

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

[67] S. Eilenberg. *Automata, Languages and Machines*. Academic Press, 1974. 2 volumes.

[68] C.C. Elgot. Monadic computation and iterative algebraic theories. In H.E. Rose and J.C. Shepherson, editors, *Logic Colloquium '73*, pages 175–230, Amsterdam, 1975. North-Holland.

[69] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier/MIT Press, 1990.

[70] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning About Knowledge*. MIT Press, Cambridge, MA, 1995.

[71] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Logic in Computer Science*, pages 193–202. IEEE, Computer Science Press, 1999.

[72] M.P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Cambridge Univ. Press, 1996.

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

[74] M.M. Fokkinga. Datatype laws without signatures. *Math. Struct. in Comp. Sci.*, 6:1–32, 1996.

[75] W.J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin, 2000.

[76] M. Forti and F. Honsell. Set theory with free construction principles. *Annali Scuola Normale Superiore, Pisa*, X(3):493–522, 1983.

[77] A.A. Fraenkel, Y. Bar-Hillel, and A. Levy. *Foundations of Set Theory*. North-Holland, Amsterdam, 2nd rev. edition, 1973.

[78] P.J. Freyd. Aspects of topoi. *Bull. Austr. Math. Soc.*, 7:1–76 and 467–480, 1972.

[79] P.J. Freyd. Recursive types reduced to inductive types. In *Logic in Computer Science*, pages 498–507. IEEE, Computer Science Press, 1990.

[80] P.J. Freyd. Algebraically complete categories. In A. Carboni, M.C. Pedicchio, and G. Rosolini, editors, *Como Conference on Category Theory*, number 1488 in Lect. Notes Math., pages 95–104. Springer, Berlin, 1991.

[81] P.J. Freyd. Remarks on algebraically compact categories. In M.P. Fourman, P.T. Johnstone, and A.M. Pitts, editors, *Applications of Categories in Computer Science*, number 177 in LMS, pages 95–106. Cambridge Univ. Press, 1992.

[82] H. Friedman. Equality between functionals. In *Logic Colloquium. Symposium on Logic held at Boston 1972 - 1973*, number 453 in Lect. Notes Math., pages 22–37. Springer, Berlin, 1975.

[83] M.J. Gabbay and A.M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Comp.*, ??:??–??, 2002.

[84] 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.

[85] J. Gibbons, G. Hutton, and Th. Altenkirch. When is a function a fold or an unfold? In A. Corradini, M. Lenisa, and U. Montanari, editors, *Coalgebraic Methods in Computer Science*, number 44 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2001.

[86] M. Giry. A categorical approach to probability theory. In B. Banaschewski, editor, *Categorical Aspects of Topology and Analysis*, number 915 in Lect. Notes Math., pages 68–85. Springer, Berlin, 1982.

[87] R. van Glabbeek. The linear time - branching time spectrum II. In E. Best, editor, *CONCUR '93. 4th International Conference on Concurrency Theory*, number 715 in Lect. Notes Comp. Sci., pages 66–81. Springer, Berlin, 1993.

[88] J.A. Goguen. Minimal realization of machines in closed categories. *Bull. Amer. Math. Soc.*, 78(5):777–783, 1972.

[89] J.A. Goguen. Realization is universal. *Math. Syst. Theor.*, 6(4):359–374, 1973.

[90] J.A. Goguen. Discrete-time machines in closed monoidal categories. I. *Journ. Comp. Syst. Sci*, 10:1–43, 1975.

[91] J.A Goguen, K. Lin, and G. Rosu. Circular coinductive rewriting. In *Automated Software Engineering (ASE'00)*, pages 123–131. IEEE Press, 2000.

[92] J.A. Goguen and G. Malcolm. A hidden agenda. *Theor. Comp. Sci.*, 245(1):55–101, 2000.

[93] 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.

[94] R. Goldblatt. *Topoi. The Categorial Analysis of Logic*. North-Holland, Amsterdam, 2nd rev. edition, 1984.

[95] R. Goldblatt. *Logics of Time and Computation*. CSLI Lecture Notes 7, Stanford, 2nd rev. edition, 1992.

[96] A.D. Gordon. Bisimilarity as a theory of functional programming. In S. Brookes, M. Main, A. Melton, and M. Mislove, editors, *Mathematical Foundations of Program Semantics*, number 1 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1995.

[97] J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification Second Edition*. The Java Series. Addison-Wesley, 2000. `http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html`.

[98] S.J. Gould. What does the dreaded "E" word mean anyway? In *I have landed. The end of a beginning in natural history*, pages 241–256. Three Rivers Press, New York, 2002.

[99] J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Inf. & Comp.*, 100(2):202–260, 1992.

[100] 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.

[101] H.P. Gumm, editor. *Coalgebraic Methods in Computer Science (CMCS'03)*, number 82(1) in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2003.

[102] H.P. Gumm, editor. *Coalgebraic Methods in Computer Science*, volume 327 of *Theor. Comp. Sci.*, 2004. Special issue on CMCS'03.

[103] H.P. Gumm, J. Hughes, and T. Schröder. Distributivity of categories of coalgebras. *Theor. Comp. Sci.*, 308:131–143, 2003.

[104] H.P. Gumm and T. Schröder. Coalgebraic structure from weak limit preserving functors. In H. Reichel, editor, *Coalgebraic Methods in Computer Science*, number 33 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2000.

[105] H.P. Gumm and T. Schröder. Products of coalgebras. *Algebra Universalis*, 846:163–185, 2001.

[106] T. Hagino. *A categorical programming language*. PhD thesis, Univ. Edinburgh, 1987. Techn. Rep. 87/38.

[107] T. Hagino. A typed lambda calculus with categorical type constructors. In D.H. Pitt, A Poigné, and D.E. Rydeheard, editors, *Category and Computer Science*, number 283 in Lect. Notes Comp. Sci., pages 140–157. Springer, Berlin, 1987.

[108] R. Hasegawa. Categorical data types in parametric polymorphism. *Math. Struct. in Comp. Sci.*, 4:71–109, 1994.

[109] I. Hasuo and B. Jacobs. Context-free languages via coalgebraic trace semantics. Techn. Rep. ICIS-R05004, Inst. for Computing and Information Sciences, Radboud Univ. Nijmegen. To appear in the LNCS proceedings of CALCO 2005., 2005.

[110] A. Heifetz and D. Samet. Topology-free typology of beliefs. *Journ. of Economic Theory*, 82(2):324–341, 1998.

[111] U. Hensel. *Definition and Proof Principles for Data and Processes*. PhD thesis, Techn. Univ. Dresden, Germany, 1999.

[112] U. Hensel and B. Jacobs. Proof principles for datatypes with iterated recursion. In E. Moggi and G. Rosolini, editors, *Category Theory and Computer Science*, number 1290 in Lect. Notes Comp. Sci., pages 220–241. Springer, Berlin, 1997.

[113] U. Hensel and B. Jacobs. Coalgebraic theories of sequences in PVS. *Journ. of Logic and Computation*, 9(4):463–500, 1999.

[114] U. Hensel and H. Reichel. Defining equations in terminal coalgebras. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent trends in Data Type Specification*, number 906 in Lect. Notes Comp. Sci., pages 307–318. Springer, Berlin, 1995.

[115] U. Hensel and D. Spooner. A view on implementing processes: Categories of circuits. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specification*, number 1130 in Lect. Notes Comp. Sci., pages 237–254. Springer, Berlin, 1996.

[116] C. Hermida. *Fibrations, Logical Predicates and Indeterminates*. PhD thesis, Univ. Edinburgh, 1993. Techn. rep. LFCS-93-277. Also available as Aarhus Univ. DAIMI Techn. rep. PB-462.

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

[118] M.W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems, and Linear Algebra*. Academic Press, New York, 1974.

[119] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985. Available at `www.usingcsp.com`.

[120] D.R. Hofstadter. *Gödel, Escher, Bach: an eternal golden braid*. Basic Books, New York, 1979.

[121] F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (co)inductive-type theory. *Theor. Comp. Sci.*, 253(2):239–285, 2001.

[122] G.E. Hughes and M.J. Cresswell. *A New Introduction to Modal Logic*. Routledge, London and New York, 1996.

[123] J. Hughes. Modal operators for coequations. In A. Corradini, M. Lenisa, and U. Montanari, editors, *Coalgebraic Methods in Computer Science*, number 44 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2001.

[124] J. Hughes. *A Study of Categories of Algebras and Coalgebras*. PhD thesis, Carnegie Mellon Univ., 2001.

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

[126] B. Jacobs. Mongruences and cofree coalgebras. In V.S. Alagar and M. Nivat, editors, *Algebraic Methodology and Software Technology*, number 936 in Lect. Notes Comp. Sci., pages 245–260. Springer, Berlin, 1995.

[127] 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.

[128] 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.

[129] B. Jacobs. Invariants, bisimulations and the correctness of coalgebraic refinements. In M. Johnson, editor, *Algebraic Methodology and Software Technology*, number 1349 in Lect. Notes Comp. Sci., pages 276–291. Springer, Berlin, 1997.

[130] B. Jacobs. *Categorical Logic and Type Theory*. North Holland, Amsterdam, 1999.

[131] B. Jacobs. Towards a duality result in coalgebraic modal logic. In H. Reichel, editor, *Coalgebraic Methods in Computer Science*, number 33 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2000.

[132] B. Jacobs. A formalisation of Java's exception mechanism. In D. Sands, editor, *Programming Languages and Systems (ESOP)*, number 2028 in Lect. Notes Comp. Sci., pages 284–301. Springer, Berlin, 2001.

[133] B. Jacobs. Many-sorted coalgebraic modal logic: a model-theoretic study. *Inf. Théor. et Appl.*, 35(1):31–59, 2001.

[134] B. Jacobs. Comprehension for coalgebras. In L. Moss, editor, *Coalgebraic Methods in Computer Science*, number 65(1) in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2002.

[135] B. Jacobs. Exercises in coalgebraic specification. In R. Crole R. Backhouse and J. Gibbons, editors, *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*, number 2297 in Lect. Notes Comp. Sci., pages 237–280. Springer, Berlin, 2002.

[136] B. Jacobs. The temporal logic of coalgebras via Galois algebras. *Math. Struct. in Comp. Sci.*, 12:875–903, 2002.

[137] B. Jacobs. Trace semantics for coalgebras. In J. Adámek and S. Milius, editors, *Coalgebraic Methods in Computer Science*, number 106 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2004.

[138] B. Jacobs. A bialgebraic review of regular expressions, deterministic automata and languages. Techn. Rep. ICIS-R05003, Inst. for Computing and Information Sciences, Radboud Univ. Nijmegen, 2005.

[139] B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors. *Coalgebraic Methods in Computer Science (CMCS'98)*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998.

[140] B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors. *Coalgebraic Methods in Computer Science*, volume 260(1/2) of *Theor. Comp. Sci.*, 2001. Special issue on CMCS'98.

[141] B. Jacobs and E. Poll. Coalgebras and monads in the semantics of Java. *Theor. Comp. Sci.*, 291(3):329–349, 2003.

[142] B. Jacobs and E. Poll. Java program verification at nijmegen: Developments and perspective. Techn. Rep. NIII-R0318, Comput. Sci. Inst., Univ. of Nijmegen., 2003.

[143] B. Jacobs and E. Poll. Java program verification at nijmegen: Developments and perspective. In K. Futatsugi, F. Mizoguchi, and N. Yonezaki, editors, *Software Security – Theories and Systems*, number 3233 in Lect. Notes Comp. Sci., pages 134–153. Springer, Berlin, 2004.

[144] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–259, 1997.

[145] B. Jacobs and J. Rutten, editors. *Coalgebraic Methods in Computer Science (CMCS'99)*, number 19 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1999.

[146] B. Jacobs and J. Rutten, editors. *Coalgebraic Methods in Computer Science*, volume 280(1/2) of *Theor. Comp. Sci.*, 2002. Special issue on CMCS'99.

[147] B. Jay. Data categories. In M.E. Houle and P.Eades, editors, *Computing: The Australasian Theory Symposium Proceedings*, number 18 in Australian Comp. Sci. Comm., pages 21–28, 1996.

[148] C.B. Jay. A semantics for shape. *Science of Comput. Progr.*, 25:251–283, 1995.

[149] P.T. Johnstone. *Topos Theory*. Academic Press, London, 1977.

[150] P.T. Johnstone. *Stone Spaces*. Number 3 in Cambridge Studies in Advanced Mathematics. Cambridge Univ. Press, 1982.

[151] P.T. Johnstone, A.J. Power, T. Tsujishita, H. Watanabe, and J. Worrell. On the structure of categories of coalgebras. *Theor. Comp. Sci.*, 260:87–117, 2001.

[152] A. Joyal and I. Moerdijk. *Algebraic Set Theory*. Number 220 in LMS. Cambridge Univ. Press, 1995.

[153] R.E. Kalman, P.L. Falb, and M.A. Arbib. *Topics in Mathematical System Theory*. McGraw-Hill Int. Series in Pure & Appl. Math., 1969.

[154] B. von Karger. Temporal algebra. *Math. Struct. in Comp. Sci.*, 8:277–320, 1998.

[155] S. Kasangian, G.M. Kelly, and F. Rossi. Cofibrations and the realization of nondeterministic automata. *Cah. de Top. et Géom. Diff.*, XXIV:23–46, 1983.

[156] P. Katis, N. Sabadini, and R.F.C. Walters. Bicategories of processes. *Journ. of Pure & Appl. Algebra*, 115(2):141–178, 1997.

[157] Y. Kawahara and M. Mori. A small final coalgebra theorem. *Theor. Comp. Sci.*, 233(1-2):129–145, 2000.

[158] S.C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, number 34 in Annals of Mathematics Studies, pages 3–41. Princeton University Press, 1956.

[159] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. & Comp.*, 110(2):366–390, 1994.

[160] M. Kracht. *Tools and Techniques in Modal Logic*. North Holland, Amsterdam, 1999.

[161] S. Krstić, J. Launchbury, and D. Pavlović. Categories of processes enriched in final coalgebras. In F. Honsell and M. Miculan, editors, *Foundations of Software Science and Computation Structures*, number 2030 in Lect. Notes Comp. Sci., pages 303–317. Springer, Berlin, 2001.

[162] C. Kupke, A. Kurz, and Y. Venema. Stone coalgebras. In H.P. Gumm, editor, *Coalgebraic Methods in Computer Science*, number 82(1) in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2003.

[163] C. Kupke, A. Kurz, and Y. Venema. Stone coalgebras. *Theor. Comp. Sci.*, 327(1-2):109–134, 2004.

[164] A. Kurz. Coalgebras and modal logic. Notes of lectures given at ESSLLI'01, Helsinki. Available as `www.cwi.nl/~kurz/??`, 1999.

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

[166] A. Kurz and J. Rosický. Operations and equations for coalgebras. *Math. Struct. in Comp. Sci.*, 15(1):149–166, 2005.

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

[168] S. Mac Lane. *Mathematics: Form and Function*. Springer, Berlin, 1986.

[169] S. Mac Lane and I. Moerdijk. *Sheaves in Geometry and Logic. A First Introduction to Topos Theory*. Springer, New York, 1992.

[170] F.W. Lawvere. *Functorial Semantics of Algebraic Theories and Some Algebraic Problems in the context of Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia Univ., 1963. Reprinted in *Theory and Applications of Categories*, 5:1–121, 2004.

[171] F.W. Lawvere and S.H. Schanuel. *Conceptual mathematics: a first introduction to categories*. Cambridge Univ. Press, 1997.

[172] 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.

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

[174] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, Berlin, 1992.

[175] K.L. McMillan. *Symbolic Model Checking*. Kluwer Acad. Publ., 1993.

[176] A. Melton, D.A. Schmidt, and G.E. Strecker. Galois connections and computer science applications. In D.H. Pitt, S. Abramsky, A. Poigné, and D.E. Rydeheard, editors, *Category Theory and Computer Programming*, number 240 in Lect. Notes Comp. Sci., pages 299–312. Springer, Berlin, 1985.

[177] T. Miedaner. The soul of the Mark III beast. In D.R. Hofstadter and D.C. Dennet, editors, *The Mind's I*, pages 109–113. Penguin, 1981.

[178] R. Milner. An algebraic definition of simulation between programs. In *Sec. Int. Joint Conf. on Artificial Intelligence*, pages 481–489. British Comp. Soc. Press, London, 1971.

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

[180] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[181] John C. Mitchell. *Foundations of Programming Languages*. MIT Press, Cambridge, MA, 1996.

[182] E. Moggi. Notions of computation and monads. *Inf. & Comp.*, 93(1):55–92, 1991.

[183] L.S. Moss. Coalgebraic logic. *Ann. Pure & Appl. Logic*, 96(1-3):277–317, 1999. *Erratum* in *Ann. Pure & Appl. Logic*, 99(1-3):241–259, 1999.

[184] L.S. Moss. Parametric corecursion. *Theor. Comp. Sci.*, 260(1-2):139–163, 2001.

[185] L.S. Moss, editor. *Coalgebraic Methods in Computer Science (CMCS'00)*, number 65(1) in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2002.

[186] L.S. Moss and I. Viglizzo. Harsanyi type spaces and final coalgebras constructed from satisfied theories. In J. Adámek and S. Milius, editors, *Coalgebraic Methods in Computer Science*, number 106 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2004.

[187] M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. PhD thesis, Univ. Nijmegen, 2004.

[188] E. Palmgren and I. Moerdijk. Wellfounded trees in categories. *Ann. Pure & Appl. Logic*, 104(1/3):189–218, 2000.

[189] 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.

[190] D. Pattinson. Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. *Theor. Comp. Sci.*, 309(1-3):177–193, 2003.

[191] D. Pattinson. An introduction to the theory of coalgebras. Course notes at the North American Summer School in Logic, Language and Information (NASSLLI), 2003.

[192] D. Pavlović and M. Escardó. Calculus in coinductive form. In *Logic in Computer Science*, pages 408–417. IEEE, Computer Science Press, 1998.

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

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

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

[196] R. Plasmeijer and M. van Eekelen. *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley, 1993.

[197] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, number 664 in Lect. Notes Comp. Sci., pages 361–375. Springer, Berlin, 1993.

[198] G.D. Plotkin. Lambda definability in the full type hierarchy. In J.R. Hindley and J.P. Seldin, editors, *To H.B Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, New York and London, 1980.

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

[200] A. Pnueli. The temporal logic of programs. In *Found. Comp. Sci.*, pages 46–57. IEEE, 1977.

[201] A. Pnueli. The temporal semantics of concurrent programs. *Theor. Comp. Sci.*, 31:45–60, 1981.

[202] E. Poll and J. Zwanenburg. From algebras and coalgebras to dialgebras. In A. Corradini, M. Lenisa, and U. Montanari, editors, *Coalgebraic Methods in Computer Science*, number 44 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2001.

[203] J. Power and D. Turi. A coalgebraic foundation for linear time semantics. In M. Hofmann D. Pavlović and G. Rosolini, editors, *Category Theory and Computer Science 1999*, number 29 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1999.

[204] H. Reichel. Third hugarian computer science conference. In *Behavioural equivalence — a unifying concept for initial and final specifications*. Akademiai Kiado, Budapest, 1981.

[205] H. Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras*. Number 2 in Monographs in Comp. Sci. Oxford Univ. Press, 1987.

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

[207] H. Reichel, editor. *Coalgebraic Methods in Computer Science (CMCS'00)*, number 33 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2000.

[208] M. Rößiger. Languages for coalgebras on datafunctors. In B. Jacobs and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 19 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1999.

[209] M. Rößiger. Coalgebras and modal logic. In H. Reichel, editor, *Coalgebraic Methods in Computer Science*, number 33 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2000.

[210] M. Rößiger. *Coalgebras, Clone Theory, and Modal Logic*. PhD thesis, Techn. Univ. Dresden, Germany, 2000.

[211] M. Rößiger. From modal logic to terminal coalgebras. *Theor. Comp. Sci.*, 260(1-2):209–228, 2001.

[212] J. Rutten. Processes as terms: non-well-founded models for bisimulation. *Math. Struct. in Comp. Sci.*, 2(3):257–275, 1992.

[213] J. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorigi and R. de Simone, editors, *Concur'98: Concurrency Theory*, number 1466 in Lect. Notes Comp. Sci., pages 194–218. Springer, Berlin, 1998.

[214] J. Rutten. Relators and metric bisimulations. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998.

[215] J. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. Technical report, CWI Report SEN-R0023, 2000.

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

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

[218] J. Rutten. A coinductive calculus of streams. *Math. Struct. in Comp. Sci.*, 15(1):93–147, 2005.

[219] 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.

[220] J.J.M.M. Rutten. Automata, power series, and coinduction: Taking input derivatives seriously (extended abstract). In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *International Colloquium on Automata, Languages and Programming*, number 1644 in Lect. Notes Comp. Sci., pages 645–654. Springer, Berlin, 1999.

[221] A. Salomaa. *Computation and Automata*, volume 25 of *Encyclopedia of Mathematics*. Cambridge Univ. Press, 1985.

[222] D. Schamschurko. Modelling process calculi with PVS. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998.

[223] O. Schoett. Behavioural correctness of data representations. *Science of Comput. Progr.*, 14:43–57, 1990.

[224] M.B. Smyth. Topology. In S. Abramsky, Dov M. Gabbai, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 641–761. Oxford Univ. Press, 1992.

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

[226] C. Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.

[227] W.W. Tait. Intensional interpretation of functionals of finite type I. *Journ. Symb. Logic*, 32:198–212, 1967.

[228] P. Taylor. *Practical Foundations of Mathematics*. Number 59 in Cambridge Studies in Advanced Mathematics. Cambridge Univ. Press, 1999.

[229] H. Tews. Coalgebras for binary methods: Properties of bisimulations and invariants. *Inf. Théor. et Appl.*, 35(1):83–111, 2001.

[230] A. Thijs. *Simulation and Fixpoint Semantics*. PhD thesis, Univ. Groningen, 1996.

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

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

[233] D. Turi and J. Rutten. On the foundations of final semantics: non-standard sets, metric spaces and partial orders. *Math. Struct. in Comp. Sci.*, 8(5):481–540, 1998.

[234] T. Uustalu, V. Vene, and A. Pardo. Recursion schemes from comonads. *Nordic Journ. Comput.*, 8(3):366–390, 2001.

[235] E.P. de Vink and J.J.M.M. Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach. *Theor. Comp. Sci.*, 221:271–293, 1999.

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

[237] M. Wand. Final algebra semantics and data type extension. *Journ. Comp. Syst. Sci*, 19:27–44, 1979.

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

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

[240] U. Wolter. CSP, partial automata, and coalgebras. *Theor. Comp. Sci.*, 280 (1-2):3–34, 2002.

[241] J. Worrell. Toposes of coalgebras and hidden algebras. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 11 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1998.

[242] J. Worrell. Terminal sequences for accessible endofunctors. In B. Jacobs and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 19 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1999.

[243] G.C. Wraith. A note on categorical datatypes. In D.H. Pitt, A. Poigné, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, number 389 in Lect. Notes Comp. Sci., pages 118–127. Springer, Berlin, 1989.