An Introduction to Coalgebra in Four Short Lectures and Two Long Appendices

preliminary handout for the participants of MGS 2018

Alexander Kurz

April 2016, revised June 2016, last revision April 23, 2018 *
Appendices not complete, way too many references still missing, further updates to follow

Abstract

Four lectures offering an introduction to coalgebra from the point of view of automata theory. The emphasis in the lectures is on examples and only a minimum of category theory is explained and used in order to clarify concepts such as coinduction and the duality of algebras and coalgebras. This is balanced by an appendix which takes the opposite approach and, assuming familiarity with basic category theory, outlines some elements of the categorical theory of coalgebras.

Contents

*Added April 23: Sections A.2, A.1.1, A.1.2,
Added April 13: Section A.9 on Kleene theorems
Added April 12: Proof of Thm 1.2, Section 3.5.2 on modal operators as natural transformations
Added April 11: Section 2.5: exercises on (co)induction
Added April 10: quick intro to modal logic in Section A.10

0 Introduction

These are the notes for a course on coalgebras that was first given at MGS 2016 and then at MGS 2018.

The course assumes that the students have seen some basic automata theory such as deterministic, non-deterministic and pushdown automata. The hope is that building on these concrete examples, one can introduce quite advanced concepts in a short series of lectures.

We aim to explain: Coalgebra, bisimulation, behavioural equivalence, final coalgebras, proof and definition by coinduction, solutions of guarded recursive equations, distributive laws, bialgebras.

In the form of slogans, some of the important ideas are

- Dynamic systems are coalgebras.
- Behaviours are what is preserved by morphisms.
- There is a system of all behaviours (the final coalgebra).
- A definition by coinduction is the arrow into the final coalgebra.
- A system of guarded recursive equations is a coalgebra.
- The unique solution to a system of guarded recursive equations is given by the unique arrow into the final coalgebra.

These notes come in four parts. The main part, Sections 1-3, explains the slogans above with examples from automata theory. Even though we provide definitions of most of the category theoretic notes involved, I hope this part will make sense without knowing category theory as it is based on concrete examples. We don't do any category theory in this part. We just use a small number of categorical notions (functor, coalgebra, natural transformation, finality, monad, distributive law) to organise the examples.

The exercise subsections at the end of each section provide some detours which can be skipped if one keeps the focus on the examples from automata theory, but are essential to more fully understand the coalgebraic generalisation of automata. They also provide additional motiviation for the following sections.

The third part, Appendix A, discusses further topics and, if time permist, will talk about some of this in lecture 4. (Let me know if you are interested in somethin in particular.)

The fourth part, Appendix B of the notes is an appendix on the category theory of coalgebras.

I am trying to write part three (overview of research topics in coalgebra) and four (some elements of the category theory of coalgebras) so that they can be attempted independently of each other and of parts one and two.

0.1 A Remark on Category Theory

I would like to comment on the role of category theory for this course and for coalgebra and computer science more generally.

Categories (and variants such as double categories, higher categories, enriched categories, etc) are mathematical structures in their own right and category theory is the area of mathematics studying them, just like group theory studies groups and lattice theory studies lattices, etc. In fact, groups, rings, posets, lattices, metric spaces, etc are not only objects in categories but are categories themselves.

In order to handle this generality, category theory invented definitions by universal property, which provide a way to 'construct' objects in a uniform way that can be instantiated in any category. For example, in the category Set of sets and functions, the product is the familiar cartesian product. In a lattice it would be meet. In other categories it might not exist. In yet other categories, it may be quite different from what we would expect of a product, for example in the category $\mathsf{Set}^{\mathrm{op}}$ dual to Set, product is disjoint union.

We see that the meaning of a construction by universal property depends on the category in which it is instantiated. The same construction can be instantiated in any category. In this sense we can think of notions such as product as *metaphors* and I believe that there is a useful analogy between the power of category theory in mathematics and the power of metaphors in language. Of course, being mathematical, these category theoretic metaphors do have a precise meaning in each context.

The example of particular importance to the course is the one of a final (or terminal) object in a category. In particular, final coalgebras will be useful as domains

- in which each element represents an equivalence class of behaviours of processes of a certain given *type*,
- in which to uniquely solve all recursive equations of certain given *format*.

The relevance of category theory for the material of this course is that it allows us to build a general theory of coalgebras for a functor (where the functor formalises what is called 'type' and 'format' above). On the other hand, once we fixed a particular type, that is, we instantiate the category theoretic definitions in a particular category, category theory as such is not strictly needed as all the reasoning can be carried out in the specific situation without reference to category theory.

This is the point of view we take in Section 1-3, due to the short time we have for the course. Concentrating on essentially one example, we do not have to rely on category theory. Nevertheless, using category theoretic terminology as metaphors is still useful, even essential as it emphasises the possible generalisations.

0.2 Acknowledgements

The idea to present a course on coalgebras based on examples from automata theory originated from discussions with Helle Hansen on one of Prakash Panangaden's legendary Barbados workshops. The 2013 workshop on Coalgebras in Computation, Logic, Probability and Quantum Physics was organised by Prakash and Bart Jacobs to whom I am indebted for a wonderfully inspiring week.

I am also indebted to all my coauthors over the years who deeply influenced my thinking on coalgebras and category theory (and, indeed, these notes are draw heavily on a joint paper with Marcello Bonsangue, Helle Hansen and Juriaan Rot [11]). Special thanks also to Larry Moss who in his breathtaking ESSLLI 1997 lectures introduced me to coalgebras and coalgebraic logic and to Jan Rutten who made it possible for me to join his wonderful coalgebra group at CWI in 2000-02.

I enjoyed enormously giving the course at MGS 2016 and the interaction with the many students (and lecturers) who took and interest, asked questions, contributed ideas for further research and helped out with improvements. I tried to incorporate what I learned from them in these notes, which were never complete during the course. So let me have another go ...

0.3 Relationship to other courses at MGS 2016/2018

Category Theory. Coalgebras are a good example to illustrate the beauty of category theory. For example, induction and coinduction in Set are not dual. Only the 'abstract' categorical duality reveals the duality between the two concepts as a formal relationship. Category theory is most useful when 'abstract' categorical concepts capture 'concrete' combinatorial phenomena. An example of this is given by the notion of bisimilarity, see Section 1.5.

Type Theory (only at MGS 2016). Many interesting types are constructed as initial algebas or final coalgebras (natural numbers, streams, all sorts of trees, ...)

Denotational Semantics and λ-calculus. The fixpoints needed in denotational semantics arise as final coalgebras. For example, for untyped lambda-calculus one would like to take the final coalgebra of $X \mapsto X^X$. Since due to 'mixed variance' this is not a functor on the category Set one needs to resort to domain theory to solve this domain equation. But if T is a functor on Set then the domain equation $X \cong TX$ always has a "largest" solution as the final T-coalgebra.

Denotational Semantics for Weak Memory Concurrency (only at MGS 2018. Coalgebra provides a general framework for structural operational semantics as bialgebras.

0.4 The course at MGS 2018, lecture by lecture.

A course allows one to follow a different path than the notes. But this path might contain some useful points of view for a reader, so here is a summary.

Lecture 1 started with an introduction why one might be interested in coalgebra. Coalgebra is a general theory of systems. (The scope of the theory will hopefully become more clear during the course.) As a general theory, if you are interested in only some particular class of systems, the general theory might still be valuable because of the conceptual explanation it provides and because of the technical results it provide. For example,

- coalgebra is the natural home for notions such as coinduction and bisimulation,
- coalgebra cuts across modal logic, domain theory, concurrency and set theory, highlighting the unity of some deep ideas that look different if viewed from only one of those areas,
- research often follows the path: take something you know, make a variation, write a paper; if the variation falls into the scope of coalgebra chances are that coalgebra will help you, maybe even with an already existing general result; many of the best papers in coalgebra prove a general result but also provide a novel instance of the general result that can be understood wihtout knowing about coalgebras and is of interest to researchers in an area such as concurrency or modal logic,
- the duality of algebra and coalgebra can only be formalised in the general theory using category theory; this becomes powerful in the theory of logics of systems when we approach this using Stone Duality; a concrete example is that coalgebra reveals that Abramsky's Domain Theory in Logical Form and Goldblatt's work on the model theory of modal logic are variations on the same basic ideas.

Lecure 1 then proceeded to cover Section 1.2.

Exercise Class 1 proved Theorem 1.2 and Exercises 1.5 and 1.6. The point of the proof of Theorem 1.2 is to show how to reduce coinduction to induction. Theorem 1.2 justifies coinduction as a definition and proof principle for languages. It also motivates the general point of view that coinduction arises from the existence of final coalgebras. We also briefly discussed that final coalgebras always exist in the sense of the Aczel-Mendler theorem.

This led to a discussion of the interplay of modal logic, concurrency and set theory, all of which discuss coalgebras $X \to \mathcal{P}X$ from different perspectives. In particular we did Exercise 1.6 showing Aczel's insight that $\mathcal{P}$-coalgebra morphisms are (functional) bisimulations in the sense of Park and Milner familiar from concurrency theory. The connection to modal logic comes from the fact that $\mathcal{P}$-coalgebra morphisms preserve the validity of modal formulas. The connection to set theory comes from the fact that the initial $\mathcal{P}$-algebras is our

familiar universe of well-founded sets and the final $\mathcal{P}$-coalgebras is Aczel's universe of non-well founded sets.

Lecture 2 defined coalgebras and discussed a range of examples, see Section A.1. We then covered Sections 2.1-2.3.

Exercise Class 2 looked at duality, induction and coinduction, see Section 2.5. We discussed with Exercise 2.10 that coinductive proofs can be seen as circular proofs in which it is allowed to use what we want to prove on arguments that are not smaller. This is closely related to the proof of Theorem 1.2, which also suggests one way of breaking the duality, namely to fit the argument into the diagram defining finality. Another solution is to work with bisimulations instead.

Lecture 3 started repeating some remarks on duality, induction and coinduction from Exercise Class 2, see Section 2.5. In particular, induction and coinduction in Set are not dual to each other and investigating how they interact is very interesting. As an example of the importance of this question to computer science, I gave a brief summary of the mathematical theory of structural operational semantics initiated by Turi and Plotkin, see Section A.7. We then proceeded to cover Sections 1.2, 2.4, 3.1.

Exercise Class 3 presented natural transformations as polymorphic functions and as modal operators, see Section 3.5. Exercise 3.16 gives a short introduction to the field of coalgebraic modal logic and also exemplifies typical techniques involved in working in coalgebra.

Lecture 4 selected some material from the appendices. While the lectures concentrated on on a specific example, I tried

- Elements of the mathematical theory of structural operational semantics (monads, comonads, distributive laws, Beck's theorem, ...) Section 3.4, A.7, B.9
- Forgetful functors of algebras/coalgebras ... adjoints, monads, comands (the other Beck's theorem) ... intersection of algebras and coalgebras ... left adjoint right adjoint ...
- Coalgebras over algebras ... Section 1.3 and 1.4
- Initial algebra sequence and final coalgebra sequence ... reducing induction to coinduction ... Barr's theorem ... Worrells' theorem

Reading order. Apart from reading through the notes sequentially, it should make sense to read in the order 1.1, 2.1-2.3, 2.4 (in conjunction with 1.2 and 2.5), 3.1, which I would consider the minimal meaningful material needed to complement the appendices.

1 Languages and Automata

In this section we review some basic automata theory from the point of view of coalgebra. We will see

- how deterministic automata are coalgebras and how the language of a state of an automaton is given by its image into the final coalgebra,
- how the final coalgebra structure interacts with the algebraic structure given by union and concatenation of languages,
- how this interaction can be used to extend our methodology to context free grammars and
- how pusdown automata fit (or not?) into the picture.

1.1 Deterministic automata

A deterministic automaton is given by a set X and an output map [1]

$$o : X \to 2$$

which maps accepting states to $1 \in 2$ and non-accepting states to $0 \in 2$ and a transition map

$$t : X \to X^A$$

which maps a state $x \in X$ to a function $t(x) : A \to X$. It will be useful in the sequel to pair the two maps o, t into one map

$$\langle o, t \rangle : X \to 2 \times X^A \tag{1}$$

and this is a safe thing to do since any map

$$\xi : X \to 2 \times X^A$$

can be written[2] as $\xi = \langle o, t \rangle$ for uniquely determined o, t.

We call such a ξ a coalgebra. Formal definition will follow in Section 2.

Now we come to the first important idea of the course: The automaton of all languages.

An important example of a deterministic automaton is the automaton of all languages.[3] It has as carrier $Z = 2^{A^*} = 2^{(A^*)}$, that is, the set of all languages, and associated maps

$$o : Z \to 2 \qquad o(z) = 1 \Leftrightarrow \varepsilon \in z \tag{2}$$

[1] For brevity, we use von Neumann's notation $2 = \{0, 1\}$.

[2] A reader familiar with category theory recognises here the universal property of the product "$\times$".

[3] We use $A^* = \coprod_{n \in \mathbb{N}} A^n$ to denote the set of all finite words over A. A *language* L is a subset of A^*, or, equivalently a map $A^* \to 2$. We write $w \in L$ or $L(w) = 1$ to say that a word w is in L.

where ε denotes the empty word and

$$t : Z \to Z^A \qquad t(z)(a) = z_a \tag{3}$$

where $z_a = \{w \in z \mid aw \in z\}$ is known as the Brzozowski language-derivative of z wrt a.

Exercise: Instantiate the definition of the automaton of all languages in case that A contains one element.

It is worth contemplating the definition of the automaton of all languages for long enough until the next theorem is obvious.

Theorem 1.1. *A state z in the automaton of all languages accepts exactly the language z.*

The above can be extended in order to describe the language accepted by any state in any deterministic automaton. This requires the notion of a (homo)morphism between deterministic automata. We define

$$f : X \to X'$$

to be a *homomorphism*, or just *morphism*, of deterministic automata iff [4]

$$o(x) = o(f(x)) \tag{4}$$
$$t(f(x))(a)) = f(t(x)(a)) \tag{5}$$

where we use the same letters o, t to denote both the structure of X and of X'. (This follows common practice in many programming languages where types are inferred from the context.)

Theorem 1.2. *For any deterministic automaton X there is a unique morphism $[\![\cdot]\!] : X \to Z$ to the automaton of all languages. Moreover, $[\![x]\!]$ is the language accepted by x.*

This property of the automaton of all languages is important and called finality.

We just met our first example of a final coalgebra, a concept we will formally define in the next Section 2.

It is possible to go directly to Section 2 from here.

[4] If we do not want to reply on type inference, we should write that the function $f : X \to X'$ is a homomorphism of deterministic automata from $(X, \langle o, t\rangle)$ to $(X', \langle o', t'\rangle)$ if

$$o'(x) = o(f(x))$$
$$t'(f(x))(a)) = f(t(x)(a))$$

Proof. We sketch the proof in the special case where $A = 1$ and $Z = 2^{\mathbb{N}}$ with o being head and t being tail. Let $\langle o', t' \rangle : X \to 2 \times X$. To give a map $X \to Z$ is to give a map $f : X \times \mathbb{N} \to 2$ which can be defined inductively

$$f(x)(0) = o(x)$$
$$f(x)(n+1) = f(t(x))(n)$$

It remains to check that these two equations are equivlent to f being a morphism of coalgebras. □

Exercise 1.3. Extend the proof above from streams to deterministic automata.

Remark 1.4.
- Note that the final coalgebra is given by a set of functions from an initial algebra.
- The idea to reduce coinduction to induction can be carried out in a different way: Often (and always in Set) the final coalgebra can be constructed as a limit of a possibly transfinite chain (the final coalgebra sequence). Then coinduction reduces to transfinite induction.

1.2 Algebraic Structure of Languages

The set of languages Z carries the structure given by o and t in (2) and in (3). In this section we discuss some important additional algebraic structure of Z, namely union and concatenation. In mathemtical jargon we say that the set Z of all languages carries the structure of an idempotent semiring

$$(Z, 0, 1, +, \times)$$

where $0 = \emptyset$ is the empty set, $1 = \{\varepsilon\}$ is the language consisting of the empty word, $+ = \cup$ is union and $\times = \cdot$ is concatenation. So we may also write

$$(Z, \emptyset, 1, \cup, \cdot).$$

Idempotence refers to $+$ being idempotent. And semiring means that $+$ is commutative and associative, $\times$ is associative, and that $0, 1$ are the respective neutral elements. Moreover, $\times$ distributes over $+$ from the left and from the right. (Z is not a ring because $+$ has no inverse.)

We are now going to show that this algebraic structure of an idempotent semiring can be defined *coinductively* in terms of the structure o, t from (1)

$$\langle o, t \rangle : Z \to 2 \times Z^A.$$

In the following we will denote elements of Z as L, M to emphasise that—despite of being states of an automaton—they also are languages. Accordingly we may write L_a for $t(L)(a)$ and $\varepsilon \in L$ for $o(L) = 1$.

With this notation we define $0, 1, +, \times$ as follows: [5]

$$\begin{array}{rl} \varepsilon \notin 0 & 0_a = 0 \\ \varepsilon \in 1 & 1_a = 0 \\ \varepsilon \in L + M \Leftrightarrow \varepsilon \in L \text{ or } \varepsilon \in M & (L + M)_a = L_a + M_a \\ \varepsilon \in L \times M \Leftrightarrow \varepsilon \in L \text{ and } \varepsilon \in M & (L \times M)_a = L_a \times M + \bar{o}(L) \times M_a \end{array} \tag{6}$$

where $\bar{o}(L)$ is just ad hoc notation to emphasize that we interprete the outputs $0, 1$ of o now as elements of the semiring Z. The meaning of the left-hand column should be obvious and the right-hand column formalises the following properties:

$\{\}$ does not contain any word (of length greater or equal to one).
$\{\varepsilon\}$ does not contain any word of length greater or equal to one.
$aw \in L + M$ iff $w \in L_a$ or $w \in M_a$.
$awv \in L \times M$ iff $w \in L_a$ and $v \in M$ or $\varepsilon \in L$ & $wv \in M_a$.

Let us emphasise, and we will explain this more formally in the next section, that it is a consequence of definition (6) that $(0, 1, +, \times)$ coincide with $(\emptyset, \{\varepsilon\}, \cup, \cdot)$.

1.3 Context-free Grammars

We write non-terminals as $x, y, \ldots$ and terminals as $a, b, \ldots$. Well bracketed expressions of the form $a^n b^n$, for example, can be derived from

$$\begin{array}{c} x \to axy \\ x \to 1 \\ y \to b \end{array} \tag{7}$$

with x as a start symbol.

Can we write 7 as a coalgebra?

Recall that in our account of languages and automata, the derivatives $(\cdot)_a$ play a crucial role, so it is essential that in the grammar above every non-empty right-hand side starts with a terminal. It is well-known that every grammar can be transformed into such a form, called the Greibach normal form. The fact that all symbols on the right-hand side, apart from the first one, are non-terminals is convenient, but not essential.

[5] Formally we should write

$$\begin{array}{rl} o(0) = 0 & t(0)(a) = 0 \\ o(1) = 1 & t(1)(a) = 0 \\ o(L + M) = o(L) \vee o(M) & t(L + M)(a) = t(L)(a) + t(M)(a) \\ o(L \times M) = o(L) \wedge o(M) & t(L \times M)(a) = t(L)(a) \times M + \bar{o}(L) \times M_a \end{array}$$

but this is less readable. Also note that the 0 is overloaded; eg the two occurrences of the symbol 0 in $o(0) = 0$ are different. The first is the 0 in $(Z, 0, 1, +, \times)$, the second is the 0 in 2 in $2 \times Z^A$. Also note that we think of 2 as the semiring (Boolean algebra) $(2, 0, 1, \wedge, \vee)$.

What is the language defined by a grammar? How do we define or compute, for example, $[\![x]\!]$? One possibility is to read both sides of $\to$ as languages, $\to$ as $\supseteq$ and then take the least solution of the inequations. Another possibility is to introduce a symbol $+$ for union and use equations

$$\begin{aligned} x &= axy + 1 \\ y &= b \end{aligned}$$

instead of inequations. For example, the first equation should really be read as

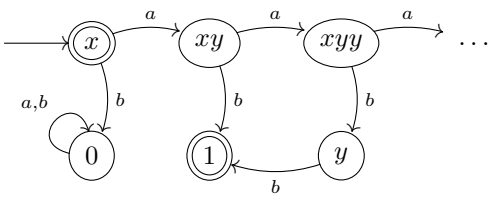
$$[\![x]\!] = [\![a]\!] \times [\![x]\!] \times [\![y]\!] + [\![1]\!] \tag{8}$$

with $[\![a]\!] = \{a\}, [\![1]\!] = \{\varepsilon\}$. So we see that the language defined by a context-free grammar comes about as the solution of a set of mutual fixed-point equations in the semiring of languages. Of course, this point of view is equivalent with the more familiar one of defining $[\![x]\!]$ as the set of words (=strings of terminals) derivable from x.

Can we recover (8) from Theorem 1.2?

1.4 Pushdown automata

There is an easy and direct way to transform a CFG such as (7) into a one-state pushdown automaton that accepts a word on empty stack. The stack symbols are the non-terminals. One starts with a stack containing only the start symbol, x in (7). Then one non-deterministically applies any matching rule. For example,



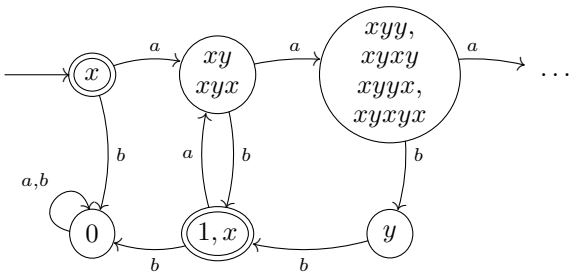
The start state is accepting because we have the rule $x \to 1$. The state labelled 1 is accepting because 1 symbolises the empty string, ie, empty stack. The state labelled 0 is a sink state.[6]

To emphasise that this way of looking at pushdown automata also works in the presence of non-determinism consider

$$\begin{aligned} x &\to axy \\ x &\to axyx \\ x &\to 1 \\ y &\to b \end{aligned} \tag{9}$$

[6] Exercise: Find the missing transitions. Extend the picture.

where, upon reading a with stack x we have two possible ways to proceed. (The second rule allows us to accept all well-bracketed expressions, not just those of the form $a^n b^n$.) Now the pushdown graph looks as follows [7]



(10)

For example, the second a-transition can be computed as follows. First, we note $(xy + xyx)_a = (xy)_a + (xyx)_a$ and we compute $(xy)_a = x_a \times y + y_a = x_a \times y = (xy + xyx) \times y = xyy + xyxy$ as well as $(xyx)_a = (xy)_a \times x = (xyy + xyxy) \times x$ so that we have

$$\begin{array}{rl} xy & \stackrel{a}{\to} xyy + xyxy \\ xyx & \stackrel{a}{\to} xyyx + xyxyx \end{array} \tag{11}$$

and, therefore,

$$xy + xyx \stackrel{a}{\to} xyy + xyxy + xyyx + xyxyx \tag{12}$$

We see that the mechanism of computing derivatives faithfully implements non-determinstic pushdown automata, but we note that the "determinisation" that happens in going from (11) to (12) looses some information. To say this more precisely, suppose we are in a pda with stack xy, then we need to know the upper row of (11) in order to continue the computation upon reading a. But the upper row of (11) cannot be recovered from (10) or (12). The inability of recovering (11) from (12) does reflect that (10) is really what could be called a determinized pushdown automaton (even though it is not a pushdown automaton, so maybe better 'determinized pushdown graph'). [8]

1.5 Exercise: Transition Systems and Bisimulation

The purpose of this extended exercise is to show that a seemingly different example exhibits the same general structure as deterministic automata. This is the example that is at the beginning of computer scientists interest in coalgebras and is discussed in detail in the monograph of Aczel on non-well founded set theory (1989).

A *transition system* (X, R), or a *Kripke frame* in modal logic, is a set X with a relation $R \subseteq X \times X$.

Let $\mathcal{P}X$ denote the set of subsets of X.

[7] Exercise: Find the missing transition. Extend the picture.

[8] Exercise: State and prove that determinization preserves the language.

Exercise 1.5. Show that transition systems (X, R) are in bijective correspondence with maps $X \to \mathcal{P}X$.

What in this example should correspond to the automaton of all languages? Theorem 1.2 shows that this automaton has the property that for each other automaton there is a unique homomorphism into it.

Taking this as our guiding idea, we seek a definition of homomorphism of transition systems. Looking back at the definition of a homomorphism of automata (4) and (5) the definition should capture the idea that if f is a homomorphism, then x and $f(x)$ have the same behaviour. Note that for $f : (X, R) \to (X', R')$ the obvious

$$xRy \;\Rightarrow\; f(x)R'f(y)$$

is not enough as this would allow $f(x)$ to have transitions (ie behaviour) that need not be reflected in the domain (X, R).

So let us try something else. Using the previous exercise, given $\xi : X \to \mathcal{P}X$ and $\xi' : X' \to \mathcal{P}X'$ we define $f : X \to X'$ to be a homomorphism if for all $x \in X$

$$f[\xi(x)] = \xi'(f(x)) \tag{13}$$

where $f[-]$ denotes direct image, that is, for all $a \subseteq X$ we have $f[a] = \{x' \mid \exists x \in X \,.\, x' = f(x) \;\&\; x \in a\}$. [9]

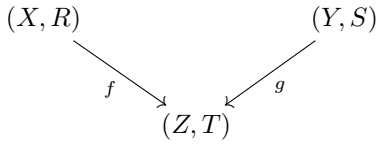
Exercise 1.6. Show that (13) is equivalent to stating that for all $x, y \in X$

$$xRy \;\Rightarrow\; f(x)R'f(y) \tag{14}$$

$$f(x)R'y' \;\Rightarrow\; \exists y \in X \,.\, xRy \;\&\; f(y) = y' \tag{15}$$

The important observation here is that in addition to the 'forward' clause (14) there is also the 'backward' clause (15). Note how both clauses correspond to the two different inclusions $\subseteq$ and $\supseteq$ of (13). Intuitively, the backward clause is needed to capture the idea that x and $f(x)$ have the same behaviour. This gives rise to the notion of (coalgebraic) behavioural equivalence.

To be more precise, given two transition systems (X, R) and (Y, S), let us say that two 'states' or 'processes' $x \in X$ and $y \in Y$ are *behaviourally equivalent* iff there are homomorphisms $f : (X, R) \to (Z, T)$ and $g : (Y, S) \to (Z, T)$



(16)

such that $f(x) = g(y)$.

[9] $f[-]$ is left-adjoint to the inverse image $f^{-1} : 2^{X'} \to 2^X$. Alternatively, one could take the right-adjoint $f[[a]] = \{x' \mid \forall x \in X \,.\, x' = f(x) \;\Rightarrow\; x \in a\}$. I am not sure the associated notion of behavioural equivalence is of interest, though.

There are two ways to think about this definition. First, we can take from general category theory that also for transition systems we an analogue of the automaton of all languages, which was shown in Theorem 1.2 to have the property that for all systems there is a unique homomorphism into it. Thus, if we take in the definition above (Z, T) to be the final transition system, then it is obvious that behavioural equivalence is indeed an equivalence relation as it is given by equality on the final system. Indeed,

behavioural equivalence on the final system is equality.

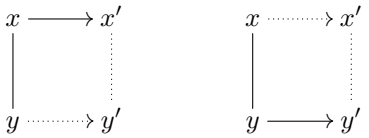
Second, in case we don't want to use the theorem of category theory that provides us with the existence of the final system, we can instead let (Z, T) range over all systems. In this case, (Z, T) may be different for different (X, R) and (Y, S) and we need to show that behavioural equivalence is transitive. [10]

Recall that a *bisimulation* between (X, R) and (Y, S) is a relation $B \subseteq X \times Y$ such that for all $x \in X, y \in Y$ we have that xBy only if

$$\forall x' \in X \,.\, (xRx' \;\Rightarrow \exists y' \,.\, ySy'' \;\&\; x'By') \tag{17}$$

$$\forall y \in Y \,.\, (ySy' \;\Rightarrow \exists x' \,.\, xRx' \;\&\; x'By') \tag{18}$$

as indicated in



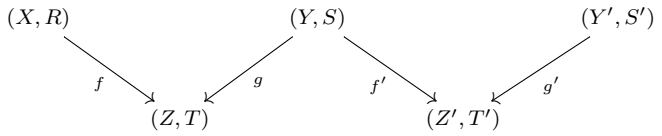
Two states are called *bisimilar* if there is some bisimulation relating them.

The next exercises show that bisimilarity and behavioural equivalence agree.

Exercise 1.7. Show that behavioural equivalence implies bisimilarity. [Hint: Given (16), show that $\{(x, y) \mid f(x) = g(y)\}$ is a bisimulation.]

Exercise 1.8. Show that bisimilar states are behaviourally equivalent. [Hint: If B is a bisimulation between (X, R) and (Y, S), then there is a relation $R_B \subseteq B \times B$ such that (B, R_B) is a transition system and such that the projections to (X, R) and (Y, S) are homomorphisms.]

[10] Consider a situation as in



We need to show that if $f(x) = g(y)$ and $f'(y) = g'(y')$ then x and y' are behaviourally equivalent in the sense of (16). To this end we build (Z'', T'') as follows. Let (Z'', T'') be the disjoint union of (Z, T) and (Z', T'') quotiented by $\{(g(y), f'(y)) \mid y \in Y\}$. Then the obvious maps $(Z, T) \to (Z'', T'')$ and $(Z', T') \to (Z'', T'')$ are homomorphisms, which gives us, by composition of homomorphisms, a picture as in (16).

Where as Kripke frames are the fundamental structure in modal logic, in computer science applications one finds more transition

Exercise 1.9. Labelled transition systems can be accounted for as maps $X \to \mathcal{P}(A \times X)$.

2 Coalgebras and Coinduction

In this section we repeat the development of the previous section using notions of category theory and coalgebra. The main purpose is to give a gentle introduction to these techniques. For example, we will see that the automaton of all languages is characterised by a universal property: The automaton of all languages is a final coalgebra. And we will exhibit the coinductive definitions of operations on languages (ie of union and concatenation) as arrows into the final coalgebra. One reason to do this is to highlight that whereas the view of automata as algebras (semigroups) and inductive definitions are widely used in automata theory, the coalgebraic and coinductive point of view is also of interest. But more importantly, as we hinted at in the exercise at the end of the last section, the coalgebraic point of view also includes (labelled) transition systems and, in fact, many more systems, including probabilistic ones.

2.1 Coalgebras

We go back to (1) where we said that a determinstic automaton is a map $\langle o, t\rangle : X \to 2 \times X^A$. To bring out the categorical and coalgebraic structure of deterministic automata, we define

$$TX = 2 \times X^A$$

and see now that we can write a determinstic automaton in the simple form

$$X \to TX.$$

This point of view will allow us to generalise from deterministic automata to other transition types T, including non-determinism, probability, two-player games, and many more.

Intuitively, a coalgebra is a map $\xi : X \to TX$ telling us the possible one-step behaviours $\xi(x)$ for a given state x. Technically, to turn this idea into a powerful general theory of systems, we need to extend the operation T from sets to functions via

$$Tf = 2 \times f^A$$

where for $f : X \to Y$ one defines

$$\begin{aligned}(2 \times f^A) : 2 \times X^A &\to 2 \times Y^A \\ (b, g) &\mapsto (b, f \circ g)\end{aligned}$$

As a first notational benefit from extending T to functions, we can now reformulate the definition of homomorphism $f : X \to X'$ between determinstic automata $\xi : X \to TX$, $\xi' : X' \to TX'$ elegantly as

$$Tf \circ \xi = \xi' \circ f$$

or, in a diagram,

$$\begin{array}{ccc} X & \xrightarrow{\xi} & TX \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle Tf} \\ X' & \xrightarrow[\xi']{} & TX' \end{array}$$

As a conceptional benefit, it makes now sense to solve equations such as

$$X = TX.$$

For example, the deterministic automaton of all languages

$$\zeta : Z \to TZ$$

is a solution of that equation (for $TX = 2 \times X^A$). But to make this precise, we need to generalise from equality $X = TX$ to isomorphism $X \cong TX$.

As a mathematical benefit, we can prove that the automaton of all languages Z is a fixed point of T in an axiomatic way that generalises to arbitrary T, thus exhibiting the abstract properties of T and Z needed to show that Z is a fixed-point of T. This is what we are going to do now: The axiomatic approach requires us to introduce the notions of category and functor, as this is exactly what is required to prove our fixed-point theorem below.

2.2 Categories and functors

We have seen the definition of a coalgebra (X, ξ) in the special case where X is a set and ξ is a function. We are going to see now that the notion of a coalgebra does not require sets and functions at all and can be formulated wrt to any category and functor.

Definition 2.1. *1. A **category** $\mathcal{C}$ consists of a class of 'objects' $X, Y, \ldots$ and for each pair of objects a set $\mathcal{C}(X, Y)$ of 'arrows'. An arrow $f \in \mathcal{C}(X, Y)$ is denoted by $f : X \to Y$. There is a binary operation of composition $\circ$ mapping $f : X \to Y, g : Y \to Z$ to $g \circ f : X \to Z$ and there are distinguished 'identity arrows' $\mathrm{id}_X : X \to X$, subject to $\circ$ being associative and* id *acting as identities wrt to $\circ$.*

*2. An **isomorphism**, or iso for short, is an arrow $f : X \to Y$ such that there is $g : Y \to X$ with $g \circ f = \mathrm{id}_X$ and $f \circ g = \mathrm{id}_Y$.* [11]

*3. A **functor** $F : \mathcal{C} \to \mathcal{C}'$ maps objects in $\mathcal{C}$ to objects in $\mathcal{C}'$ and arrows in $\mathcal{C}(X, Y)$ to arrows in $\mathcal{C}'(FX, FY)$ so that identities and composition are preserved.*

[11] We say that two objects X, Y are isomorphic if there exists an isomorphism $X \to Y$.

4. *A* ***coalgebra*** *for a functor $T : \mathcal{C} \to \mathcal{C}$ is an arrow $X \to TX$ in $\mathcal{C}$. A coalgebra-morphism $f : (\xi : X \to TX) \to (\xi' : X' \to TX')$ is an arrow $f : X \to X'$ in $\mathcal{C}$ such that $Tf \circ \xi = \xi' \circ f$. The category with coalgebras as objects and coalgebra-morphisms as arrows is denoted by* $\mathsf{Coalg}(T)$.

Our main example is where $\mathcal{C}$ is the category Set of sets and functions and $\mathsf{Coalg}(T)$ is the category of deterministic automata. There are at least three reasons to introduce the notion of coalgebra in the above generality.

- This axiomatic approach reveals exactly what is needed to prove the fixed point theorem (known as Lambek's Lemma). Moreover, and some of it is explained in the appendix, quite a large part of a general theory of systems can be developed in such an axiomatic style.
- As it will turn out below, pushdown automata are coalgebras over a category different than the category Set of sets and functions. In Section **??** and the appendix we will see many more examples of systems that arise as coalgebras over other base categories than Set.
- The exercises in Section **??** it reveals the duality of algebras and coalgebras or, more precisely, that coalgebras over Set are dual to algebras over the dual of Set.

A good way of learning category theory, and sometimes also of discovering new thoerems of category theory, is to look at categories as generalisations of posets or lattices.

Example 2.2. Let $\mathcal{C}$ be a preorder, or poset, or lattice. [12] Then $\mathcal{C}$ is a category of a special kind with at most one arrow between any two objects. (And, conversely, all categories of this kind are preorders.) A functor is just a monotone operation. And a coalgebra $X \to TX$ is a post-fix point $X \leq TX$.

It is easy to show that the largest post-fix point (if it exists) is a fixpoint by looking at the following diagram (read $\to$ as $\leq$ again)

$$\begin{array}{ccc} X & \longrightarrow & TX \\ \uparrow & & \uparrow \\ TX & \longrightarrow & TTX \end{array}$$

where the lower horizontal arrow comes from monotonicity of T and the vertical arrows come from $X \to TX$ being the largest postfixpoint.

If $\mathcal{C}$ is a poset, then the diagram above gives us $X = TX$. If $\mathcal{C}$ is only a preorder, then we only get that X is equivalent, or isomorphic, to TX.

The reason we detailed the argument that the largest postfixpoint is a fixpoint is that the proof of the fixpoint theorem in the next section will be a direct

[12] A preorder is a set with a symmetric and transitive relation. A poset is a preorder where the relation is anti-symmetric. A lattice is a poset with finite joins and meets.

generalisation. This also serves as an example of the general methodology of discovering category theoretic proofs by generalising the corresponding lattice theoretic ones.

2.3 Final coalgebras

Coming back to deterministic automata, recall that Theorem 1.2 revealed that the automaton Z of all languages enjoys what is called a universal property: For any automaton X there is a unique homomorphism to Z. We turn this property into a definition. Note that this definition makes sense in any category.

Definition 2.3. *An object Z in a category is called final, or terminal, if for any object X there is a unique arrow $X \to Z$.*

As a first indication of the power of the axioms of category one proves the easy but crucial

Proposition 2.4. *If a category has a final object, then it is determined 'up to canonical isomorphism'. That is, if Z, Z' are final, then the 'canonical' arrows $Z \to Z'$ and $Z' \to Z$ are isomorphisms.* [13]

The proposition is important as it tells us that, up to isomorphism, the final coalgebra is uniquely determined by its universal property. This provides us with many advantages. For example, as discussed in Appendix B.4, it is known for general categorical reasons that the final T-coalgebra exists. Moreover, it defines a notion of 'behavioural equivalence'. Theorem 1.2 stating that the automaton of all languages is the final T-coalgebra (for $TX = 2 \times X^A$) then becomes a representation theorem giving an explicit description of the object defined abstractly (up to isomorphism) by the property of being final in $\mathsf{Coalg}(T)$. [14] Theorem 1.1 stating that a state z in the automaton of all languages accepts the language z then confirms that the 'final coalgebra semantics'

$$[\![\cdot]\!] : X \to Z$$

assigns to a state x in an automaton indeed the language accepted by x.

We are now in a position to prove the fixed-point theorem, known as Lambek's lemma. Statement and proof are a direct generalisation of the well-known fact that the largest post-fixed point of a monotone operator on a complete lattice is a fixed point. Here it is worth noting that a complete lattice (as any poset) is a category, with a functor then being a monotone operator and a coalgebra being a post-fixed-point. [15]

[13] A universal property always says that 'there exists a unique arrow' with a certain property. The expression 'canonical arrow' always refers to this unique arrow, leaving the universal property it arises from to the context.

[14] It might be interesting to show a different presentation, namely the final DA of streams, cf. Def 7.2 and Ex. 7.3 of the SDE survey. See also section 3.1 of Helle's paper with Clemens, Jan and Joost on k-regular sequences.

[15] Exercise: Prove this statement. Hint: First prove it in the case that T is a monotone operation on a proset or lattice. Then generalise the proof from posets to categories.

Theorem 2.5 (Lambek's Lemma)**.** *Let T be a functor on a category $\mathcal{C}$. If the final coalgebra $\zeta : Z \to TZ$ exists then ζ is an isomorphism.*

2.4 Coinductive definitions

In its simplest form a coinductive definition defines a map $C \to Z$ into a final coalgebra Z by giving a map $C \to TC$

$$\begin{array}{ccc} C & \longrightarrow & TC \\ \downarrow & & \downarrow \\ Z & \longrightarrow & TZ \end{array} \tag{19}$$

Then $C \to Z$ is uniquely determined by $C \to TC$ and finality of $Z \to TZ$.

Example 2.6. Let $TX = 2 \times X^A$ and recall that the final coalgebra Z is the automaton of all languages. We want to show that

$$\begin{array}{rl} \varepsilon \notin 0 & 0_a = 0 \\ \varepsilon \in 1 & 1_a = 0 \\ \varepsilon \in L + M \Leftrightarrow \varepsilon \in L \text{ or } \varepsilon \in M & (L+M)_a = L_a + M_a \end{array}$$

from Section 1.2 are coinductive definitions. For the first one, we take $C = 1 = \{*\}$ and let $h : \{*\} \to T\{*\}$ be given by $h(*) = (0, \lambda a.*)$.

$$\begin{array}{ccc} 1 & \xrightarrow{h} & T1 \\ {\scriptstyle \emptyset}\downarrow & & \downarrow{\scriptstyle T(\emptyset)} \\ Z & \xrightarrow[\langle o,t \rangle]{} & TZ \end{array} \tag{20}$$

To check that this indeed defines 0, we need to check that (20) commutes, where we write $\emptyset : 1 \to Z$ for the function that maps $*$ to the empty language. This is indeed the case and follows by a simple (but tedious if spelled out in all detail) unrolling of the respective definitions.

We leave the coinductive definition of 1 given by an h fitting into

$$\begin{array}{ccc} 1 & \xrightarrow{h} & T1 \\ {\scriptstyle 1}\downarrow & & \downarrow{\scriptstyle T(1)} \\ Z & \xrightarrow[\langle o,t \rangle]{} & TZ \end{array} \tag{21}$$

as an exercise.

For the last one we take $C = Z \times Z$ and $C \to TC$ becomes

$$Z \times Z \xrightarrow{h} 2 \times (Z \times Z)^A \tag{22}$$

$$(L, M) \mapsto (\varepsilon \in L \cup M \ , \ \lambda a.(L_a, M_a)) \tag{23}$$

Note how h arises from a direct transcription of

$$\varepsilon \in L + M \Leftrightarrow \varepsilon \in L \cup M \qquad (L + M)_a = L_a + M_a$$

Again checking that

$$\begin{array}{ccc} Z \times Z & \xrightarrow{h} & TZ \\ {\scriptstyle +}\downarrow & & \downarrow{\scriptstyle T(+)} \\ Z & \longrightarrow & TZ \end{array} \tag{24}$$

commutes establishes that $+$ is defined coinductively by h.

After having gained some experience with the exercise above, we would expect

$$\varepsilon \in L \times M \Leftrightarrow \varepsilon \in L \text{ and } \varepsilon \in M \qquad (L \times M)_a = L_a \times M + o(L) \times M_a$$

to fall into the same schema, but any straightforward attempt will fail, because the definition of $\times$ requires the use of $+$, which cannot be accommodated directly by the format (19). This has to wait a bit ...

2.5 Exercise: Duality, (co)algebras and (co)induction

In category theory we have a duality principle. Suppose we proved a theorem stating that property P holds for all categories. Then P also holds for all opposite categories. So we can reformulate P in terms of opposite categories and get another theorem.

For example, dualising Definition 2.3, we define an object to be **initial** in $\mathcal{C}$ if it is final in $\mathcal{C}^{\mathrm{op}}$. To highlight typical uses of duality do

Exercise 2.7. Convince yourself that

- an object A is initial if for all objects B there is a unique arrow $A \to B$,
- the dual of Proposition 2.4 says that any two initial objects are unique up to canonical isomorphism,
- the dual of Theorem 2.5 says that the structure of an initial algebra is an isomorphism.

In particular, Theorem 2.5 and its dual imply that if $T : \mathsf{Set} \to \mathsf{Set}$ is a functor then both initial T-algebras and final T-coalgebras are isomorphisms. Note that these two statements are not dual to each other, as coalgebras over Set are not dual to algebras over Set but dual to algebras over $\mathsf{Set}^{\mathrm{op}}$. On the other hand, the dual of Theorem 1.2 seems less useful as it turns a simple fact about deterministic automata in Set into another statement in $\mathsf{Set}^{\mathrm{op}}$ that is equivalent but less intuitive.

To better appreciate that induction and coinduction are only dual 'in the abstract' but not dual 'in the concrete' context of a specific category, it is worth to look at [7] and at [53]. Barr [7] shows that under typical assumptions on T the final T-coalgebra is the metric completion of the initial algebra. Worrell [53] shows that while the initial algebra sequence of the powerset functor $\mathcal{P}$ converges after ω steps, the final coalgebra sequence only converges after $\omega + \omega$ steps.

The following exercise in reading diagrams shows that (the basic form of) induction amounts to the initiality of the natural numbers.

Exercise 2.8. The algebra $[0, S] : 1 + \mathbb{N} \to \mathbb{N}$ is the initial algebra for the functor $TX = 1 + X$.[16] Show that the diagram

$$\begin{array}{ccc} 1 + X & \xrightarrow{z,s} & X \\ {\scriptstyle 1+f}\uparrow & & \uparrow{\scriptstyle f} \\ 1 + \mathbb{N} & \xrightarrow{[0,S]} & \mathbb{N} \end{array} \tag{25}$$

commutes if and only if

$$f(0) = z \tag{26}$$

$$f(Sn) = s(f(n)) \tag{27}$$

Remark 2.9. Initiality of the natural numbers, via Diagram (25), directly only gives iteration (26) and (27). More complicated forms of induction or recursion can be proved by combining (25) with other type constructors available in Set. Ultimately, all forms of induction and recursion over the natural numbers can be justified their initiality.

The solution for the folowing exercise can be found click here.

Exercise 2.10. Let $TX = A \times X$. The final T-coalgebra is

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

Give coinductive definitions of $\mathit{zip} : A^{\mathbb{N}} \times A^{\mathbb{N}} \to A^{\mathbb{N}}$, $\mathit{even} : A^{\mathbb{N}} \to A^{\mathbb{N}}$ and define $\mathit{odd}(l) = \mathit{even}(\mathit{tail}(l))$. Prove by coinduction

$$\mathit{zip}(\mathit{even}(x), \mathit{odd}(x)) = x$$

for all $x \in A^{\mathbb{N}}$ and justify your proof with the finality of $A^{\mathbb{N}}$ (which in turn was proved in Theorem 1.2). □

[16] To explain the notation, $0 : 1 \to \mathbb{N}$ is the map that maps the element of 1 to the natural number 0 and $S : \mathbb{N} \to \mathbb{N}$ is the successor function. In the same way, the constant map z picks an element of X and s is a function $X \to X$. $+$ denotes coproduct (disjoint union) and $[0, S]$ and $[z, s]$ are the maps that makes the obvious case distinctions. Similarly, the map $1 + f$ makes a similar case distinction, with 1 in $1 + f$ denoting the identity map on 1. (It is a convenient abuse of notation to denote by the same symbol an object and its identity arrow.)

3 Algebraic and coalgebraic structure

The aim of this section is threefold. First, we want to solve the problem we left at the end of last section, namely how to account for the definition of $\times$ coinductively. Second, we will reveal that the coinductive definition arises from an intricate interplay of algebraic and coalgebraic structure that has many interesting instances and about which we can prove powerful general theorems. Third, we show how to solve guarded recursive equations in the final coalgebra.

All of this will be achieved by bringing into the picture the category theoretic notion of a monad. In fact, for our exposition, we will not need the precise definition of a monad (see the appendix) as we will use only one particular monad and we will use it mainly as a notational device (or metaphor, as explained in the introduction) in order to simplify and clarify our exposition. The notion of a monad is then required to say precisely how far the methods of this seciton generalise and to prove general theorems making this method more powerful.

This section can also be seen as an introduction by one example to the mathematical theory of structural operational semantics initiated by Turi and Plotkin. See Section A.7 for more on this.

3.1 Coinductive definitions, continued

Let us go back to the problem of solving the equation

$$(L \times L')_a = L_a \times L' + o(L) \times L'_a$$

via the existence of a unique arrow into the final coalgebra. The problem was that starting with a pair (L, L') in the upper left corner in the diagram

$$\begin{array}{ccc} Z \times Z & \xrightarrow{h} & TZ \\ {\scriptstyle \times}\downarrow & & \downarrow{\scriptstyle T(\times)} \\ Z & \longrightarrow & TZ \end{array} \tag{28}$$

the left-hand side $(L \times L')_a$ of the equation corresponds to going down and right in the diagram and the right-hand side $L_a \times L' + o(L) \times L'_a$ should correspond to going right and down. But with the format of the diagram, we can only represent right-hand sides that use a $\times$ and o and t, but not any of the other algebraic operations such as $+$.

Therefore we are going to collect all terms that can be formed from the semiring operations in one operation M that maps a set X to the set MX for all semiring terms over X modulo the idempotent semiring equations.

Such an M is a monad, but we do not need to know this right now.

Instead of writing three separate Diagrams (20), (21), (24), we can now write just one and even capture $\times$:

$$\begin{array}{ccc} MZ & \xrightarrow{h} & TMZ \\ {\scriptstyle [\![-]\!]}\downarrow & & \downarrow{\scriptstyle TM([\![-]\!])} \\ Z & \longrightarrow & TZ \end{array} \tag{29}$$

The function $[\![-]\!]$ we want to define coinductively, that is by specifying h, is evaluation of semiring terms containing languages as additional constants. For example, we want that $[\![(0+1)\times(1\times L)]\!] = L$ for any $L \in Z$, or that $[\![\{a\}\times\{b\}]\!] = \{ab\}$.

Technically, we need that M is a functor. That is easy. Given $f : X \to Y$, we define $Mf : MX \to MY$ simply by taking a term t in MX and replacing all occurrences of some $x \in X$ in t by $f(x)$.

Next, we need to check that definition (6), repeated here for convenience,

$$\begin{array}{rl} \varepsilon \notin 0 & 0_a = 0 \\ \varepsilon \in 1 & 1_a = 0 \\ \varepsilon \in L + L' \Leftrightarrow \varepsilon \in L \text{ or } \varepsilon \in L' & (L + L')_a = L_a + L'_a \\ \varepsilon \in L \times L' \Leftrightarrow \varepsilon \in L \text{ and } \varepsilon \in L' & (L \times L')_a = L_a \times L' + \bar{o}(L) \times L'_a \end{array} \tag{30}$$

does specify a map $h : MZ \to TMZ$. To this end, denote the components of h by o and t so that we have $h = \langle o, t\rangle$ and let us write $\langle o^Z, t^Z\rangle$ for the structure of the final coalgebra. With this notation (30) becomes, with $l, l' \in MZ$,

$$\begin{array}{rl} o(0) = 0 & t(0)(a) = \emptyset \\ o(1) = 1 & t(1)(a) = \emptyset \\ o(l + l') = o(l) \vee o(l') & t(l + l')(a) = t(l)(a) + t(l')(a) \\ o(l \times l') = o(l) \wedge o(l') & t(l \times l')(a) = t(l)(a) \times l' + \bar{o}(l) \times t(l')(a) \end{array} \tag{31}$$

which, together with a base case for elements $L \in Z$ [17]

$$o(L) = o^Z(L) \qquad t(L)(a) = t^Z(L)(a) \tag{32}$$

is an inductive definition over the structure of semiring terms of the map

$$MZ \xrightarrow{h=\langle o,t\rangle} TMZ, \tag{33}$$

which is the map which in turn provides the coinductive definition of the desired interpretation $MZ \to Z$ of semiring terms as operation on the set Z of all languages.

To summarize, we have seen how the notation of a monad M allows us account for more complicated coinductive definitions that use arbitrary algebraic terms on the right hand side.

[17] Recall that $Z \subseteq MZ$ and that MZ is the set of terms over Z, so to define a function from MZ we need to define it on elements of Z and on the operations of the signature.

We also would like to emphasize that (31) is both an inductive definition of $MZ \to TMZ$ as well as a coinductive definition of $MZ \to Z$.

There is one technical complication we didn't talk about. Namely that MZ is the set of semiring terms over Z quotiented by the idempotent semiring equations. To finish our development we have to show that (33) is a semiring morphism, that is, that (31) preserves the equations of idempotent semirings. This is not difficult to verify: Use that the semiring structure of $TMZ = 2 \times MZ^A$ is given by the fact that both 2 and MZ^A are idempotent semirings.

Exercise 3.1. Show that (33) is indeed a semiring homomorphism.

Exercise 3.2. Show that the function $[\![-]\!]$ in (29) defined coinductively via (31) satisfies, as intended,

$$\begin{array}{rcl} [\![0]\!] & = & \emptyset \\ [\![1]\!] & = & \{\epsilon\} \\ [\![l + l']\!] & = & [\![l]\!] \cup [\![l']\!] \\ [\![l \times l']\!] & = & [\![l]\!] \cdot [\![l']\!] \end{array} \tag{34}$$

[Hint: By the uniqueness of an arrow into the final coalgebra, it is enough to show that the function $[\![-]\!]$ defined by (34) does make (29) commute.]

Let us quickly step back and take stock of what we have done. We have seen that the semiring structure on the automaton Z can be defined coinductively and we have justified the schema of coinductive definition by using the universal property of Z as a final coalgebra. Of course, the obvious inductive definition of the semiring structure given by (34) is easier than the coinductive one given by (31). So what have we gained?

Maybe not so much if we only look at this particular example. But broadening the perspective we have gained a lot because the same principle of coinductive definitions carries over to coalgebras of all type functors and applies to coinductive definitions that do not have an obvious description in terms of set-theoretic operations such as $\cup$ and $\cdot$. Moreover, notice that the inductive definition (34) only works in case we choose, among all isomorphic final coalgebras, the particular one which has the set of all languages as its carrier. On the other hand, the coinductive definition (31) is purely in terms of the coalgebra structure and independent of any particular set-theoretic presentation of the final coalgebra.

Furthermore, in the Section 3.3 we will see that the same technique will allow us to solve guarded recursive equations in the final coalgebra.

3.2 Coalgebras over algebras

[This section can be skipped. But it illustrates some interesting points that will be mentioned later again.]

We have equipped the final coalgebra Z with the structure of a idempotent semiring using coinductive definitions. But taking a closer look at (31), we see

that there is more important structure hidden. Namely, (31) tells us that not only Z but also TZ carries the structure of a semiring and that $Z \to TZ$ is a semiring morphism.[18] This is a very important observation as it shows that the final semantics is compositional wrt semiring operations.

To see that (31) implies that $TZ = 2 \times Z^A$ carries the structure of a semiring, let us write (b, l) for an element of TZ. Then we define

$$\begin{array}{rcl} 0 & = & (0, \lambda a.\emptyset) \\ 1 & = & (1, \lambda a.\emptyset) \\ (b,l) + (c,m) & = & (b \vee c, \lambda a.l(a) \cup m(a)) \\ (b,l) \times (c,m) & = & (b \wedge c, \lambda a.l(a) \cdot M \cup c \cdot m(a)) \end{array} \tag{35}$$

where $0, 1$ in the left-hand column refer to the defined operations on TZ and in the right-hand column they refer to the elements of the Boolean algebra 2 of truth-values. M is an abbreviation for the language given by (c, m), that is,

$$M = c \cdot 1 \cup \bigcup_{a \in A} \{a\} \cdot m(a).$$

In $c \cdot m(a)$ we abuse notation and identify $0, 1 \in 2$ with $0 = \emptyset$ and $1 = \{\epsilon\}$ in Z.

Of course, to substantiate the claims made in this subsubsection one needs to solve

Exercise 3.3. Check that with the definitions of (31) and (35), the final coalgebra structure $Z \to TZ$ becomes a semiring homomorphism.

3.3 Solutions of guarded recursive equations

In Section 1.3 on context-free grammars, we described the language given by a context free grammar (in Greibach normal form) in terms of equations

$$\begin{array}{rcl} x & = & a \times x \times y + 1 \\ y & = & b \end{array} \tag{36}$$

or, shorter,

$$\begin{array}{rcl} x & = & axy + 1 \\ y & = & b \end{array} \tag{37}$$

Such equations are called guarded recursive, where guarded refers to the left-most symbol on the right-hand side not being one of the recursion variables.

Already in the book by Barwise and Moss (1996), it is noted that a system of such equations corresponds to a coalgebra. As they are interested in solving equations such as

$$x = \{x\} \tag{38}$$

[18] Categorically speaking, T can be lifted to a functor on idempotent semirings and $Z \to TZ$ then can be considered as a coalgebra not over sets but over idempotent semirings.

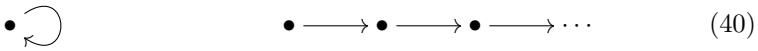
a system of equations is simply a coalgebra

$$X \to \mathcal{P}X$$

where $\mathcal{P}$ is the powerset functor.[19] Furthermore, such systems of equations have unique solutions $[\![\cdot]\!] : X \to Z$ in the final coalgebra given by

$$\begin{array}{ccc} X & \longrightarrow & TX \\ {\scriptstyle [\![\cdot]\!]}\downarrow & & \downarrow \\ Z & \longrightarrow & TZ \end{array} \tag{39}$$

For example, with $X = \{x\}$ and the equation (38) we have that $[\![x]\!]$ is the non-well founded set that can be pictured as a loop or also as an infinite list [20]

 (40)

The case of context-free languages is more complicated because the right-hand sides may involve arbitrary terms formed from the signature of a semiring. To explain this in more detail note that the right-hand side of (38) only involves $\{\cdot\}$ and that $\{\cdot\}$ – denoting set-formation – comes from $\mathcal{P}$, that is, T itself. On the other hand, the right hand sides of (37) do not only involve o and t, but also $0, 1, +, \times$. That is, we are really looking at systems of equations of the kind

$$X \to 2 \times MX^A$$

where MX is as in Section 3.1 the set of semiring terms over X modulo the idempotent semiring equations.

For example, (37) corresponds to the map $\langle o, t\rangle : X \to 2 \times MX^A$ given by

$$\begin{array}{lllll} X = \{x, y\} & o(x) = 1 & t(x)(a) = xy & t(x)(b) = 0 \\ & o(y) = 0 & t(y)(a) = 0 & t(y)(b) = 1 \end{array} \tag{41}$$

where we abbreviate $x \times y$ by xy.

Exercise 3.4. What is the map $X \to 2 \times MX^A$ corresponding to (9)?

Above, we said that our situation is more complicated than the one discussed by Barwise and Moss, as we have terms in MX on the right-hand side of the equations. Nevertheless, it is still the case that all guarded recursive equations of the form $X \to 2 \times MX^A$ have unique solutions in the final coalgebra of all languages.

[19] Some readers may worry about the existence of a final coalgebra for the powerset functor, but this is not a problem. One can either restrict to finite subsets or allow the carrier of the final coalgebra to be a proper class. Either works fine.

[20] The answer to the question why these two presentations are equivalent is that they are bisimilar.

Let us try to find out what the general picture behind this is. First, it looks like we are in some trouble: We want solutions in the final coalgebra

$$Z \to TZ$$

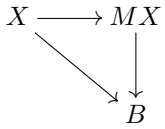
but our guarded recursive equations are of the form

$$X \to TMX.$$

Of course, if we could extend $X \to TMX$ to $MX \to TMX$, we would have a coalgebra and, hence, a unqiue morphism $MX \to Z$ and then also $X \to MX \to Z$.

How can we obtain $MX \to TMX$ from $X \to TMX$?

Fact 3.5. *Let B be an idempotent semiring. Then any function $X \to B$ extends to a semiring morphism $MX \to B$ uniquely determined by making*



commute.

(In order to check the claim, one defines $MX \to B$ by extending the map $X \to B$ to terms over X, noting that terms over X modulo the axioms of an idempotent semiring are in 1-1 correspondence with elements of M.)

Remark 3.6. The fact above is often formulated by saying that MX is (the carrier of) the free semiring over X. More generally, for any monad M it is the case that MX is (the carrier of) the free algebra over X.

We see from the fact that to answer our question, it is enough to equip TMX with the structure of an idempotent semiring. But, wait, haven't we done something like this in (35)?

To see that (6) implies that $TMZ = 2 \times MZ^A$ carries the structure of a semiring, let us write (b, f) for an element of TMZ. Then we define

$$\begin{array}{rcl} 0 & = & (0, \lambda a.\emptyset) \\ 1 & = & (1, \lambda a.\emptyset) \\ (b, f) + (c, g) & = & (b \vee c, \lambda a.f(a) + g(a)) \\ (b, f) \times (c, g) & = & (b \wedge c, \lambda a.f(a) \times m + c \times g(a)) \end{array} \tag{42}$$

where $m = c \times 1 + \sum_{a \in A} a \times g(a)$.

Let us summarise. We have recast in categorical terminology how the production rules of a CFG correspond to guarded recursive equations and how their solution is computed by the unique arrow into the final coalgebra. To do this,

we had to recognise that the final coalgebra is a coalgebra internal in the category of idempotent semirings. This corresponds to the fact the semantics is compositional, that is, for example,

$$[\![axy + 1]\!] = [\![a]\!] \cdot [\![x]\!] \cdot [\![y]\!] \cup 1$$

Note that this is indeed the crucial property in the definition of 'the language of a CFG': One computes the language of, say, xy by computing the language $[\![x]\!]$ of x, the language $[\![y]\!]$ of y and then concatenating to $[\![xy]\!] = [\![x]\!] \cdot [\![y]\!]$.

We have also seen that for a context free grammar $X \to TMX$, the corresponding automaton

$$MX \to TMX$$

can be seen as a recogniser for the language of the context free grammar, see Section 1.3.

3.4 Distributive laws and bialgebras

We have seen that the coinductive definition (6) gives rise to a rich structure: An algebra structure $MZ \to Z$ on the final coalgebra Z, as well as on TZ and TMZ, and a coalgebra structure $MZ \to TMZ$ on the algebra MZ.[21] We also have seen that $Z \to TZ$ is an algebra morphism, showing that the final semantics of the algebraic operations is compositional, a feature that is at the core of denotational semantics. In this section, we will sketch how all of these properties and more arise from a simple category theoretic idea, namely that of a distributive law, which are natural transformations of a certain kind.

Definition 3.7. *Given categories $\mathcal{C}, \mathcal{D}$ and functors $F, G : \mathcal{C} \to \mathcal{D}$, a natural transformation $\tau : F \to G$ is a family $\tau_C : FC \to GC$ indexed by objects of $\mathcal{C}$ such that $Gf \circ \tau_C = \tau_{C'} \circ Ff$ for all arrows $f : C \to C'$ in $\mathcal{C}$.*

The definition is best remembered as a commuting diagram

$$\begin{array}{ccc} FC & \xrightarrow{\tau_C} & GC \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle Gf} \\ FC' & \xrightarrow{\tau_{C'}} & GC' \end{array}$$

Intuitively, this condition means that τ only depends on F and G but is invariant under transforming the index C, see Section **??** for more details.

We will now show that the coinductive defintion (6) gives rise to a distritubive law. First, in order to emphasize that the relationship between distributive laws and structural operational semantics is a general one, we now use notation common in the structural operational semantics of process algebras as follows:

[21] Recall that MZ in our example was the free idempotent semiring over Z.

process algebraic sugar	coalgebra $\langle o, t\rangle : X \to 2 \times X^A$
$x\Uparrow$	$o(x) = 0$
$x\Downarrow$	$o(x) = 1$
$x \xrightarrow{a} y$	$t(x)(a) = x'$

$\bar{o}(x)$ again abbreviates the side condition "if $x\Downarrow$ then 1 else 0""

We now rewrite (6) in yet a another form.

$$\frac{}{0 \xrightarrow{a} 0} \tag{43}$$

$$\frac{x\Downarrow}{(x+y)\Downarrow} \qquad \frac{y\Downarrow}{(x+y)\Downarrow} \qquad \frac{x \xrightarrow{a} x' \qquad y \xrightarrow{a} y'}{x + y \xrightarrow{a} x' + y'} \tag{44}$$

$$\frac{}{1\Downarrow} \qquad \frac{}{1 \xrightarrow{a} 0} \tag{45}$$

$$\frac{x\Downarrow \qquad y\Downarrow}{(x \times y)\Downarrow} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \times y \xrightarrow{a} x' \times y + \bar{o}(x) \times y'} \tag{46}$$

This notation emphasises the reading of the coinductive defintion (6) in terms of transition relations. For example, we read (44) as saying that the 'process' $x+y$ succedes if either x or y do, and, moreover, as showing how the a-successor of $x + y$ arises from the a-successors of x and y.

The aim of this section is that the above can be succinctly summarised as a distributive law

$$MT \to TM$$

or a slight variation thereof. The mathematical benefit obtained from this is explained in more detail in Appendix **??**.

Union of languages, coinductively

The purpose of the next exercise is to show that (43) and (44) can be understood as defining a natural transformation

$$MT \to TM$$

where now M is the monad of semi-lattices generated by operation $(0, +)$ and T is is as before the functor for deterministic automata:

Exercise 3.8. Let $TX = 2 \times X^A$ as before and let $\mathcal{P}_\omega$ be the finite powerset functor, that is, $\mathcal{P}_\omega X$ is the set of finite subsets of X. We can identify elements

of $\mathcal{P}_\omega X$ with terms over the signature $(0, +)$ modulo the equations of a semi-lattice.[22] Recalling from ... the definition of TX and $\mathcal{P}_\omega X$ on maps, show that

$$\begin{array}{rcl} \mathcal{P}_\omega(2 \times X^A) & \xrightarrow{\tau_X} & 2 \times (\mathcal{P}_\omega X)^A \\ \{(b_i, l_i) \mid i \in I\} & \mapsto & (\bigvee\{b_i \mid i \in I\}, \lambda a.\{l_i(a) \mid i \in I\}) \end{array} \tag{47}$$

is a natural transformation.[23]

Next we need to convince ourselves that (47) indeed captures the clauses for 0 and + from (6), or equivalently from (43) and (44), which were formalised in Section 2.4 using the unique arrows into the final coalgebra $Z \to TZ$ given by (20) and (24).

To this end we check that the diagram below, with $M = \mathcal{P}_\omega$ and $TX = 2 \times X^A$ and $\bigcup : MZ \to Z$, commutes.

$$\begin{array}{ccccc} MZ & \longrightarrow & MTZ & \xrightarrow{\tau_Z} & TMZ \\ \vdots & & & & \vdots \\ \downarrow & & & & \downarrow \\ Z & & \longrightarrow & & TZ \end{array} \tag{48}$$

In more detail: The lower horizontal map in the diagram is given by Z being a final coalgebra. The upper horizontal map contains the coinductive definition given by applying M to the final coalgebra structure and post-composing with the distributive law $\tau : MT \to TM$ from (47). By finality, this determines a unique arrow $MZ \to Z$. To conclude that this $MZ \to Z$ is given by union, it is enough to check that with $\bigcup : MZ \to Z$ Diagram (48) commutes:

Exercise 3.9. Show that Diagram (48) commutes. In detail: Starting with a finite set of languages $S = \{L_i \mid i \in I\}$ in the upper left-hand corner of Diagram (48), show that $\langle o, t\rangle \circ \bigcup(S) = T(\bigcup) \circ \tau_Z \circ \mathcal{P}_\omega(\langle o, t\rangle)(S)$. Decomposing $\langle o, t\rangle$ this amounts to checking

$$\begin{aligned} o(\bigcup S) &= \bigvee\{o(L_i) \mid i \in I\}, \\ t(\bigcup S)(a) &= \bigcup\{t(L_i)(a) \mid i \in I\}. \end{aligned}$$

[22] To be really precise, there are three monads involved here: $S \to S/{\equiv} \to \mathcal{P}_\omega$, where S is the monad of all terms generated by operations $(0, +)$ and $S/{\equiv}$ is the monad of these terms quotiented by the equations of semi-lattices. The natural transformation $SX \to \mathcal{P}_\omega X$ maps 0 to the empty set and + to union. In our present discussion we confuse but simplify matters by largely ignoring S and not distinguishing between $S/{\equiv}$ and $\mathcal{P}_\omega$. For more details, see Appendix **??**.

[23] If we represent sets as semi-lattice terms the same can be written as

$$\begin{aligned} \tau : \mathcal{P}_\omega T &\to T\mathcal{P}_\omega \\ 0 &\mapsto (0, \lambda a.0) \\ (b, l) + (c, m) &\mapsto (b \vee c, \lambda a.l(a) + m(a)) \end{aligned}$$

To summarize, we have seen an example of how a coinductive definition of algebraic operations M on T-coalgebras corresponds to a natural transformation

$$MT \to TM.$$

Having seen union of languages, we will continue with a more involved example, namely concatenation.

Concatenation of languages, coinductively

To account for the definition of $\times$, the simple scheme needs to be complicated somewhat. Skipping the next exercise does not disturb the flow of the overall development.

The purpose of the next exercise is to show that $\times$ fits into the slightly more general form

$$M(X \times TX) \to TMX$$

The intuition here is that in a distributive law of the kind $MT \to TM$, the T-semantics of an expression such as $x+y$ does depend only on the T-semantics of x and y but not on x and y themselves. On the other hand, in (46)

$$x \times y \xrightarrow{a} x' \times y + \bar{o}(x) \times y'$$

we see a y on the right hand side of $\xrightarrow{a}$.

Exercise 3.10. Continuing from Exercise 3.10, let MX be the set of terms over X built from operations $(0, 1, +, \times)$ modulo the equations of an idempotent semiring. Show that

$$\begin{array}{rcl} M(X \times TX) & \xrightarrow{\tau_X} & TMX \\ 0 & \mapsto & (0, \lambda a.0) \\ (x,b,l) + (y,c,m) & \mapsto & (b \vee c, \lambda a.l(a) + m(a)) \\ 1 & \mapsto & (1, \lambda a.0) \\ (x,b,l) \times (y,c,m) & \mapsto & (b \vee c, \lambda a.l(a) \times y + b \times m(a)) \end{array} \tag{49}$$

is a natural transformation.

We need to check that the τ of (49) indeed defines on languages the operations $(\emptyset, \{\epsilon\}, \cup, \cdot)$. To this end we check that the diagram below, with M as in the exercise above and $TX = 2 \times X^A$ and $[\![-]\!] : MZ \to Z$ given by $(\emptyset, \{\epsilon\}, \cup, \cdot)$ commutes.[24]

$$\begin{array}{ccccccc} MZ & \longrightarrow & M(Z \times Z) & \longrightarrow & M(Z \times TZ) & \xrightarrow{\tau_Z} & TMZ \\ {\scriptstyle [\![-]\!]}\vdots & & & & & & \vdots{\scriptstyle TM[\![-]\!]} \\ Z & & \longrightarrow & & & & TZ \end{array} \tag{50}$$

[24] In detail, $[\![-]\!]$ is defined inductively via $[\![z]\!] = z$ for $z \in Z$ and $[\![0]\!] = \emptyset$, $[\![1]\!] = \{\epsilon\}$, $[\![x + y]\!] = [\![x]\!] \cup [\![y]\!]$, $[\![x \times y]\!] = [\![x]\!] \cdot [\![y]\!]$. The purpose of the exercise is to show that this inductive definition coincides with the coinductive one given by the final coalgebra.

(The unlabelled arrows are to be labelled in the obvious way: $Z \to TZ$ is the final coalgebra structure, $Z \to Z \times Z$ is the 'diagonal' $z \mapsto (z, z)$, with the upper row applying M to these arrows as appropriate.)

Exercise 3.11. Show that Diagram (50) commutes. In detail: By induction on the structure of terms in MZ, we have 4 cases: for 0, for 1, for $x + y$ and for $x \times y$. In each case the commutativity is established by chasing the respective terms through the diagram. For example, starting with $x + y$ in the top-left, we find that both paths through (50) map $x + y$ to $(o([\![x]\!]) \vee o([\![y]\!]), \lambda a.t([\![x]\!])(a) \cup t([\![y]\!])(a))$. [25]

To summarize, we used notation from category theory to organise some well-known notions of formal languages and automata theory:

- behaviour can be encoded in a functor T
- coinductive definitions arise from arrows into the final T-coalgebra (the automaton of all languages)
- algebraic operations and equations can be encoded in a functor M(such functors M have the special property of being a monad, which is discussed more fully in the appendix)
- packaging up algebraic operations in one functor M and coalgebraic operations ('observsevers', 'destructors') in one functor T gives many notational advantages which we exploited for example in Diagrams (19), (29), (48), (50)
- most notably, we saw that our example of a coinductive definition can be brought in to the simple form of a natural transformation

$$MT \to TM$$

Apart from giving us notational benefit, we saw that category theory also provided us with some useful results:

- there is a unique homomorphism into the final coalgebra
- all final coalgebras are isomorphic
- the structure of the final coalgebra is an isomorphism

I would like to finish this introduction with another example. Namely, if one has a distributive law $MT \to TM$, then

- initial algebra and final coalgebra semantics coincide

[25] Use that by definition of how M acts on arrows we have that $Mf(0) = 0$ and $Mf(1) = 1$ and $Mf(x + y) = Mf(x) + Mf(y)$ and $Mf(x \times y) = Mf(x) \times Mf(y)$ and use the definition of $[\![-]\!]$ of Footnote 24.

- bisimilarity is a congruence

This can be seen from the following diagram

$$\begin{array}{ccccc} MZ & \longrightarrow & Z & \longrightarrow & TZ \\ \uparrow & & \uparrow & & \uparrow \\ MI & \longrightarrow & I & \longrightarrow & TI \end{array} \tag{51}$$

$MI \to I$ is the initial M-algebra, $Z \to TZ$ is the final T-coalgebra. We have seen how algebra structure $MZ \to Z$ arises from the distributive law via coinduction. One obtains $I \to TI$ in a dual manner. Now there are two arrows $I \to Z$, one given by initiality of I and the other by finality of Z. But, in fact, these two arrows have to be the same, that is, initial and final semantics coincide. Moreover, this arrow is the largest bisimulation on I (because Z is final) and is a congruence (because it is an algebra morphism).

Much more structure arises from distributive laws, see Appendix B.9.

3.5 Exercise: Natural transformations

In the previous section, we have seen that a structural operational semantics appears as a natural transformation. In this section we try to get a better intuition for natural transformations and collect more examples: Polymorphic functions and even modal operators are natural transformations.

One point we will make is that if you want to characterise explicitly the natural transformations $F \to G$ between two given functors F, G then try the Yoneda lemma.

3.5.1 Polymorphic functions

We think of functors as type constructors. Then there is only one polymorphic function from the identity-functor to the identity-functor, namely the identity function:

Exercise 3.12. Let Id be the identity functor $\mathsf{Set} \to \mathsf{Set}$. There is only one natural transformation $\mathrm{Id} \to \mathrm{Id}$. [Hint: Conclude from

$$\begin{array}{ccc} 1 & \xrightarrow{\tau_1} & 1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f} \\ X & \xrightarrow{\tau_X} & X \end{array}$$

that τ_X is the identity.]

This generalises to the fact that the only 'polymorphic' functions $A \times X \to B \times X$ are those which act on the parameters according to some function $A \to B$ and are the identity on X:

Exercise 3.13. Show that there are exactly two natural transformations $\mathrm{Id} \to \mathcal{P}$.

Exercise 3.14. On the category Set, consider the functors $FX = A \times X$ and $GX = B \times X$. Show that the natural transformations $F \to G$,

$$A \times X \to B \times X,$$

are in bijection with maps $A \to B$.

The term 'polymorphic' comes from programming: As shown in the exercise, the transformations $A \times X \to B \times X$ natural in X are those functions that can be programmed without knowing the datatype X, that is, they are polymorphic in X. This correspondence carries over to more complicated examples, as we are going to explore now.

The next question is what are the natural transformations

$$X \times X \to X \times X$$

Our intuition is that natural transformations have to act as the identity on X, but here they can depend on the position of the elements of X, so that there are exactly 4 natural transformations given by, respectively,

$$\begin{aligned}(x_1, x_2) &\mapsto (x_1, x_2)\\ (x_1, x_2) &\mapsto (x_2, x_1)\\ (x_1, x_2) &\mapsto (x_1, x_1)\\ (x_1, x_2) &\mapsto (x_2, x_2)\end{aligned}$$

This example suggests that natural transformations do not change the values x_1, x_2 but may change their 'position'.

A generalisation of the above example (note that $X \times X \cong X^2$) is the following

Exercise 3.15. On the category Set, consider the functors $FX = X^A$ and $GX = X^B$. Show that the natural transformations $F \to G$,

$$X^A \to X^B,$$

are in bijection with maps $B \to A$.

This last exercise can be solved either by explicit combinatorial considerations, or by using a simple but powerful category theoretic device called the Yoneda lemma, see Appendix B.7.

3.5.2 Modal operators

Modal operators $\Box$ and $\Diamond$ can be seen as providing a logic of $\mathcal{P}X$ and this can be extended recursively to all modal formulas as in Definition A.4.

If modal logic (with empty set of atomic propositions) is the logic of coalgebras

$$X \to \mathcal{P}X$$

it is natural to ask what is the modal logic of a T-coalgebra

$$X \to TX.$$

For more on this question see Section A.10.

For the purposes of the following, we will not distinguish between a modal operator such as $\Box$ or $\Diamond$ and its semantics. The semantics of a modal operator, given a coalgebra $\xi : X \to TX$ is a function

$$2^X \to 2^X.$$

What should be a modal operator for T-coalgebras? Recall that 2^- is a contravariant functor, so we can look at and factor the semantics of a modal operator $2^X \to 2^X$ through the inverse image $2^\xi = \xi^{-1}$ of the coalgebra structure:

$$2^X \xleftarrow[2^\xi]{} 2^{TX} \xleftarrow[\lambda_X]{} 2^X \tag{52}$$

For each such λ and $x \in X$ we can define the semantics of a modal operator $[\lambda]$

$$x \Vdash [\lambda]\phi \;\Leftrightarrow\; x \in (2^\xi \circ \lambda_X)(\phi) \tag{53}$$

Natural transformations $2^X \to 2^{TX}$ are called (unary) predicate liftings.

Exercise 3.16.

1. For $TX = \mathcal{P}X$ defining $\lambda_X(\phi) = \{a \subseteq X \mid a \subseteq \phi\}$ we have
$$x \Vdash [\lambda]\phi \;\Leftrightarrow \xi(x) \subseteq \phi$$
which is just the usual semantics of the $\Box$ operator.

2. Check that the λ_X above is natural.

3. Define the λ that corresponds to $\Diamond$.

4. Using the Yoneda lemma show that (unary) predicate liftings are in bijective correspondence with 2^{T2}.

5. List all (unary) predicate liftings for $T = \mathcal{P}$. Can you show that they correspond (up to logical equivalence) to all formulas of modal logic that can be written with formulas in one free propositional variable p and where every occurrence of p is under the scope of exactly one modal operator?

6. Show that if a λ as in (52) is natural then the semantics of $[\lambda]$ according to (53) is invariant under T-bisimilarity.

This exercise gives a good flavour of working in coalgebra. The definition (52) of a modal operator via predicate liftings works in any category for which we have a contravariant functor 2^-. The semantic (53) assumes that T is a functor on Set but various generalisations to other concrete categories suggest themselves. In these categories, naturality allows us to recover, but now for arbitrary T, the property that we presented in Section A.10 as a central observation of modal logic (item 6 above). The proof of item 6 uses some easy manipulation of diagrams. On the other hand, the proof of item 5 is more difficult and exemplifies that categorical properties may be correspond to complicated combinatorial results the proof of which requires familiarity with the subject are (here modal logic) and cannot rely on typical category theoretic techniques.

A Further topics and pointers to the literature

So far we have used category theory only to clarify a few concepts. Some of the benefits achieved were:

- The coalgebraic view on automata suggested the importance of the coalgebra of all languages.
- The automaton of all languages is the final coalgebra, which justifies proofs and definitions by coinduction.
- Choosing the final coalgebra as our semantic domains allows us to solve all guarded recursive equations uniquely.
- The interplay of algebraic and coalgebraic structure necessary for interesting coinductive definitions and recursive equations, can be formulated succinctly in terms of certain natural transformations, called distributive laws,

  $$MT \to MT$$

  if we are willing to use coalgebras for a functor T and algebras for a monad M.

We have also seen a small number of category theoretic results that gave us confidence to be on the right track. For example,

- Final coalgebras are unique up to isomorphism.
- Largest fixpoints of a monotone operation on a poset are a special case of final coalgebras. Dually, least fixpoints are initial algebras.
- The structure of an initial algebra or a final coalgebra is an isomorphism.

While I hope that these observations are interesting in their own right, their main purpose is not to teach us something about automata theory. Their purpose is to illustrate concepts of astonishing generality.

Accordingly, two driving forces of the research on coalgebraic methods in computer science are

- generalising and axiomatizing, trying to find the most general assumptions under which certain results on coalgebras can be established,
- specialising coalgebraic methods to particular examples.

The first emphasises mathematical theorems, the second applications to computer science. The beauty of the subject is in the interplay of the two. We will see examples of this in the following subsections.

A.1 Probabilistic transition systems and other examples

We list some examples that have been influential in our understanding of coalgebras. Next to the examples we have seen in the lectures, various probabilistic transition systems stand out as one success story of coalgebra.

A.1.1 Coalegbras generalising automata

The simplest example of coalgebras are coalgebras

$$X \to A \times X,$$

that is, coalgebras for the functor $TX = A \times X$. The behaviour of these coalgebras can be understood as streams (infinite lists) over A. While they are very simple, they already reveal a lot of interesting structure and have been studied in detail by Rutten and collaborators eg in [39, 40, 42, 43, 26]. This line of research also studies the interplay of algebraic and coalgebraic structure of which Section 3 gave an example. The plan to study streams and then generalise has also been exploited in the introductory course [].

The main example of these notes

$$X \to 2 \times X^A,$$

deterministic automata, is only slightly more complicated than streams, now involving also input and not only output. It also goes back to Rutten, see [38], and many variations have been studied eg in [].

A.1.2 Coalegbras generalising transition systems

From the point of view of modal logic and also of concurrency and set theory, the paradigmatic example of coalgebras is

$$X \to \mathcal{P}X$$

or also

$$X \to \mathcal{P}(A \times X)$$

for transition systems labelled in A.[26] The importance of this example is not only that it cuts across modal logic, set theory and concurrency, but also that exhibits bisimulation as a coalgebraic notion, see Sections 1.5 and A.5 and B.3.

Coalgebra provides a general theory of bisimulation, uniform over a wide variety of examples. One of the most successful directions of research in coalgebra has been to take results from modal logic/concurrency and generalise it to coalgebras. This line of research is already visible in Aczel's book on Non-well founded set theory [3] and the paper by Aczel-Mendler on A final coalgebra theorem [4].

[26] Note that $\mathcal{P}(A \times X) \cong (\mathcal{P}X)^A$ so that in this example we can think of A as input or as output.

This work brought to light that $\mathcal{P}$-coalgebra-morphisms capture bisimularity, gave a coalgebraic definition of T-bisimularity, and proved an important theorem on T-coalgebras, namely the existence of final coalgebras, without any special assumptions on T.

Of course, once you start getting new general results, it becomes interesting to look for more examples that fall into the scope of the general theory.

Probabilistic transition systems come in many variations. A simple but important idea is to replace the powerset functor $\mathcal{P}$ with the distribution functor $\mathcal{D}$ which is defined so that $\mathcal{D}X$ is the set of finitely supported probability distributions.[27] The final coalgebra for $\mathcal{D}$ is not interesting because, intuitively, there are no interesting observations that can be made.[28] The comparison with $\mathcal{P}$-coalgebras suggests to consider variations such as $1 + \mathcal{D}$ or $\mathcal{D}(A \times \mathrm{Id})$.[29] The possibly earliest (?) example of this research direction is by Vink and Rutten [15, 16] showing that coalgebraic bisimulation arising from the distribution functor $\mathcal{D}$ coincides with the classic notion from concurrency. Other examples include [**?**, 10, 27, **?**]

Probabilistic transition systems on infinite state spaces lead to consider coalgebras on topological spaces or measurable spaces. Two monographs on this very interesting topic are [].

A.1.3 Coalgebras beyond transition systems

topological spaces with open maps
- neighborhood frames
- monotone neighbourhood frames
- conditional logic
- ...

A.2 Composing Functors

The previous section, Section A.1, emphasised a variety of basic functors $\mathsf{Set} \to \mathsf{Set}$ of interest to coalgebra. It is important to notice that from these basic functors we can build an infinite collection of potentially interesting functors by composition.

For example, it is of interest to mix non-determinism with probabilism. A prominent example are Segala systems [] and relatives []. These systems were originally studied in concurrency as independent variations, but can be unified

[27] $\mathcal{D}X = \{d : X \to [0,1] \mid \sum_{x \in X} d(x) = 1$ and $d(x) = 0$ except for finitely many $x \in X\}$. On functions $(\mathcal{D}f)(d)(y) = \sum_{f(x)=y} d(x)$ for $f : X \to Y$. Exercise: Show that $\mathcal{D}$ is a functor (even a monad).

[28] Exercise: The final $\mathcal{D}$-coalgebra is the one element coalgebra. Contrast this with final $\mathcal{P}$-coalgebras.

[29] $\mathcal{D}_{\leq}X = \{d : X \to [0,1] \mid \sum_{x \in X} d(x) \leq 1$ and $d(x) = 0$ except for finitely many $x \in X\}$ is ismorphic to $1 + \mathcal{D}$.

as coalgebras for functors that all arise from different ways of composing the functors $\mathcal{P}$ and $\mathcal{D}$ ... [13, 14]

Defining classes of interesting functors by basic functors and closing under composition has been an important tool in domain theory (see eg [1]) and functional programming (see eg the discussion on strictly positive functors in Lecture 4 of Venanzio Capretta's MGS lecture on λ-calculus).

Indeed, inductively defined classes of functors also play a role in coalgebra, for example the class of Kripke polynomial functors [37, 36, 28].

But a specific coalgebraic point of view on systems is to investigate T-coalgebras either by proving results for aribrary functors T or by axiomatically restricting the class of functors T. Early examples of this line of research are Rutten's Universal Coalgebra discussed in the next section and Moss's coalgebraic logic, see Section A.10.1 (and in both works, restricting to weak pullback preserving functors plays a big role).

In some sense, this shift of perspective from the properties of basic functors to the properties of classes of functors explains much of the differences between the areas of domain theory and coalgebra (another being that much, but certainly not all, of coalgebra is concerned with functors on sets whereas domain theory is concerned with functors on ordered and metric structures).

The shift to classes of functors is motivated mathematically by clearly stating the minimal assumptions needed to prove specific results. But also from a more practical point of view, the shift to simpler base categories such as Set increased the number of interesting examples of functors enough so that it becomes desirable to have results formulated generally enough so that they are robust against adding more basic functors into the class of "interesting" ones.

A.3 Universal Coalgebra

Universal algebra is the area of mathematics which generalises examples of algebras such as monoids, groups, rings, lattices, Boolean algebras, etc to algebras given by a operations and by equations (aka generators and relations). A number of interesting results, such as the isomorphism theorems or Birkhoff's characterisation of varieties (ie equationally definable classes of algebrasas) as so-called HSP classes can be proved at that level of generality. More recently universal algebra became important in solving some outstanding questions about constraint satisfaction problems.

Universal coalgebra [41] investigates properties of dynamic systems at a similar level of abstraction by dualising algebras to coalgebras. Coalgebraic duals to some theorems of universal algebras, such as the isomorphism and HSP theorems can be established [19, 18, 6]. The proposal that model logics 'dualise' equational logic was made in [33, 32, 31].

Beyond these basic results of universal algebra, the theory of universal algebra and universal coalgebra diverge. Roughly speaking the reason is that in both

cases one is (not exclusively but) mostly intersted in algebras over Set and coalgebras over Set and Set is not self-dual. In other words, properties that distinguish between Set and $\mathsf{Set}^{\mathrm{op}}$ are responsible for the differences between universal algebra and universal coalgebra (after all, coalgebras over Set are algebras over $\mathsf{Set}^{\mathrm{op}}$.)

For example, in universal algebra, lattices of congruences and Malcev terms play an important role. A category theoretic account of some this can be found in [17]. The classes of categories studied in [17] (such as Malcev, protomodular and semi-abelian) are typically not closed under dualisation and are not immediately relevant to coalgebras over Set.

On the other hand, for coalgebras over Set the notion of bisimilarity (or behavioural equivalence or coalgebraic equivalence or observational equivalence as we also have called it) plays an important role and it does not have a counterpart in universal algebra. We will discuss this in more detail in Section **??**.

An important line of investigation in universal coalgebra consists in relating properties of functors T with properties of T-coalgebras. For example, Rutten's original [41] contains a number of important results linking properties of the type functor T with properties of bisimulations. For instance, if T preserves weak pullbacks then the composition of bisimulations is a bisimulation. In a series of papers [21, 24, 23, 20, 25, 22], Gumm and T. Schröder showed that typically the converse of these results holds as well. For instance, if the composition of bisimulations is a bisimulation, then T preserves weak pullbacks.

Another important line of investigation asks which of these results can be extended to other base categories than Set. Often, one would assume that the base category is locally finitely presentable, see eg [5], and that the functor T is finitary.

A.3.1 Universal Algebra

Ideas taken from universal algebra inspired a lot of the early work on universal coalgebra. For example, Rutten showed that the well-known isomorphism theorems also hold in universal coalgebra. And it was soon shown that also Birkhoff's variety theorem can be dualised to (different variations of a) covariety theorem []. But after these initial successes, it became clear that differences between the two subjects are certainly larger than initially thought. Nevertheless, there could still be something to discover here.

I want to add some remarks on the differences between universal algebra and universal coalgebra.

bisimulation, malcev, borceux/bourn, ...

A.3.2 Domain Theory

As remarked above, the technical differences to domain theory, at least in the early stages of coalgebra, have been the focus on classes of functors on Set.

With this comes also another shift in emphasis, namely away from denotational semantics of programming languages to models of computation. Whereas many developments in domain theory started from looking at programming languages (such as the lambda calculus, PCF, ...), in coalgebra one more often likes to start with the coalgebras and only ask then what a programming language for them would be. [30]

While these differences may lead to technically quite different tools and methods, from a slightly wider point of view it seems appropriate to think of both areas as part of the same subject.

In particular, *order-enriched and metric-space-enriched coalgebra* is interested in essentially the same structures as domain theory. But the emphasis on general functors may bring some new results, see eg [].

To close the circle of ideas in this section on universal coalgebra, work on order-enriched and metric-space-enriched coalgebra, in order to build modal logics via duality, also raises interest in *ordered-enriched and metric-space-enriched universal algebra*, a topic in which there are still novel results to achieve [] and which deserves further exploration.

A.4 Final coalgebras

The importance of final coalgebras was well-known in domain theory as final not only are solutions to domain equations but, typically, are the desired solution. But as in many categories of domains initial algebras and final coalgebras coincide, and since initial algebras where more familiar at the time, it took some time for the importance of final coalgebras to become widely recognized. As far as I can judge, the turning point was Aczel's book Non-well founded set theory [] and the paper [].

The book starts from the question of how to give a semantics to Milner's Calculus of Communicating Systems (CCS). Aczel uses the ideas of domain theory, but notices that the relevant domain equation

$$X \cong \mathcal{P}(A \times X)$$

can be solved in the category of sets if one is willing to work in a category where proper classes are admitted as objects.[31] Actually, in the book, he shows that in the category of non-well founded sets the class of all non-well founded sets Z solves the equation

$$Z = \mathcal{P}Z$$

up to equality. There are a few publications on solving domain equations up to equality, but while elegant if possible, this typically introduces unnecessary

[30] But, of course, many classic texts in domain theory such as the Compendium [] put the mathematical perspective first as well.

[31] The real obstacle to solving domain equations are mixed variance operations such as $X \mapsto X^X$. These do require the techniques invented by Dana Scott [], see also [2].

complications. For example, it makes solutions dependent on set-theoretic foundations and 'implementation details' that should not be relevant.[32] Moreover, the isomorphisms

$$Z \cong TZ$$

have computational meaning. $TZ \to Z$ is sometimes called 'fold' and $Z \to TZ$ is called 'unfold'. In particular, from the coalgebraic point of view, the isomorphism $Z \to TZ$ represents the computational behaviour of the final coalgebra. This is particularly obvious in case of the final coalgebra

$$\langle head, tail \rangle : A^\omega \to A \times A^\omega$$

mapping a stream to 'head' and 'tail'.

A.4.1 Existence of final coalgebras

Due to Cantor's diagonal argument there can be no $X \cong \mathcal{P}X$ in Set. There are the following solutions to the problem.

First, restrict the cardinality of the subsets. For example, replace $\mathcal{P}$ by $\mathcal{P}_\omega$ given by $\mathcal{P}_\omega(X)$ as the set of finite subsets of X.

Second, work in a category that also admits proper classes. Define $\mathcal{P}(X)$ as the class of subsets of X.

Third, assume that there is an inaccesible cardinal κ and replace by $\mathcal{P}$ by $\mathcal{P}_\kappa$ given by $\mathcal{P}_\kappa(X)$ as the set of subsets of X of cardinality $< \kappa$.

The first solution restricts attention to finitely branching transition systems and is the easiest if it covers all relevant examples. The second and third are essentially equivalent, see also the discussion in [].

The trick in item three has the advantage of giving final coalgebras for completely general reasons. Indeed, it is easy to verify that the forgetful functor U from coalgebars to sets preserves colimits. The cardinality restriction then makes sure that the solution-set condition of Freyd's adjoint functor theorem is satisfied and hence a right F of U must exist. The final coalgebra is then given by $F1$.

But these general consideration are not quite enough to give the Aczel and Mendler theorem, which ensures that the cardinality of $F1$ is $\leq \kappa$.

A.4.2 Final coalgebra sequence

The final coalgebra of the Aczel-Mendler theorem (or of the adjoint functor theorem, see MacLane [] for the proof), is given by a large colimit: The coproduct over all coalgebras quotiented by the largest bisimulation.

[32] For instance, whether $X = A \times X$ can be solved up to equality may depend on how '$\times$' is defined. On the other hand, whether $X \cong A \times X$ can be solved only depends on the universal property of product and the category one is working in.

This (non-constructive) construction as a quotient of a large coproduct does not always give us enough information about the structure of the final coalgebra. When this happens, it is worth trying to construct the final coalgebra as a limit of a chain. Moreover, this is often a small limit, for example in case of finitary functors on Set.

For example, ... streams ... powerset ...

A.4.3 Further topics

Rational fixed-points ...

A.5 Bisimulation and Coinduction

Bisimulation and Coinduction appeared independently to various degrees in modal logic, computer science and set theory. For the history of the subject consult [**?**, Sections 3-5].

From our point of view, see Section **??**, both bisimulation and coinduction arise from the category theoretic principle of a unique arrow into the final coalgebra.

A.5.1 Coinduction

While the dual idea, namely how induction arises from initial algebras, has been in the mainstream at least since Lawvere's definition of a natural number object [], I believe that coinduction became a recognised method of proof and definition only after the work of Aczel []. The basic ideas are spelt out, for example, in Barwise and Moss, Moss, Jacobs and Rutten, Rutten and Turi.

A.5.2 Bisimilarity, Bisimulations

Bismilarity on a coalgebra $X \to TX$ is given by the kernel of the unique arrow into the final coalgebra. Categorically, this relation arises as a pullback ... and it is easy to see that if T weakly preserves pullbacks than the relation R is indeed a bisimulation in the sense of Aczel-Mendler [].

Definition A.1 (Aczel-Mendler bisimulation)**.** ...

A nice point about this definition is that it is easy to see that it coincides with the classical definition in case that $T = \mathcal{P}$ is the powerset functor.

This classical definition is powerful because it is local. Maybe it is not surprising that the notion of coalgebra is general enough to admit cases that do not allow such a neat caractersation of bisimilarity. In particular, in order to establish nice properties of bisimulations (bisimulations are closed under unions and composition, the largest bisimulation exists, etc) one needs to assume that T weakly preserves pullbacks (=preserves weak pullbacks).

It was noted in [**?**] that if we replace spans by cospans, the definition of bisimulation is better behaved and the assumption that T preserves weak pullbacks

can be dropped. An example of this situation occurs for the functor $T = 2^{2^-}$. A careful study of bisimulation for this functor was carried out in [].

Maybe the easiest definition of bisimilarity (or behavioural equivalence as it is sometimes called in situations where one does not want to assume that bisimulations with good properties exist) is the following. As far as I know it was first made explicit in [**?**]

Definition A.2. *...*

This definition has two advantages. First, it uses no extra structure than the arrows of $\mathsf{Coalg}(T)$ for an arbitrary functor $T : \mathsf{Set} \to \mathsf{Set}$. Second, it is immediate from the definition that the equivalence classes modulo bisimilarity are the connected components of the so-called category of elements of the forgetful functor $U : \mathsf{Coalg}(T) \to \mathsf{Set}$. And it is, indeed, easy to show

Proposition A.3. *Let $T : \mathsf{Set} \to \mathsf{Set}$ be a functor. There is a final coalgebra in $\mathsf{Coalg}(T)$ iff $U : \mathsf{Coalg}(T) \to \mathsf{Set}$ has a colimit.*

Proof. We only indicate that if U has a colimit Z, then Z carries the structure of a coalgebra (since TZ is the vertex of a cocone over U). Moreover, existence of an arrow into this coalgebra structure is immediate and uniqueness ... □

... tbc ...

A.5.3 Bisimulation via Relation Lifting

It was noted by Park and Milner that bisimilarity is a greatest fixed point of a monotone operator on a lattice of relations. (Of course, this is dual to the familiar explanation of inductive definitions as smallest fixed points.) One way to formalise this idea for coalgebras is to extend the functor $T : \mathsf{Set} \to \mathsf{Set}$ to a functor

$$\overline{T} : \mathsf{Rel} \to \mathsf{Rel}$$

where Rel is the category of sets as objects and binary relations as arrows. Rel is order-enriched, which means that homsets are partial orderes given by inclusion. The appropriate notion of a functor $\mathsf{Rel} \to \mathsf{Rel}$ then is that of a *locally monotone functor*, or enriched functor, that is, a functor that preserves the order on homsets. The fundamental theorem here, see Barr [], Hermida [], etc is that a functor $T : \mathsf{Set} \to \mathsf{Set}$ can be extended to a locally monotone functor $\overline{T} : \mathsf{Rel} \to \mathsf{Rel}$ iff T preserves weak pullbacks. In this case then, bisimilarity is given as the largest fixed point of the operator

...

... tbc ...

A.5.4 Further topics and references

If you need a detailed comparision of different notions of bisimulation consult Sam Staton's []

If you want to see more practical examples of coinduction in different contexts the following may be of interest (if you know more references I should add please let me know).

Classic papers on relation lifting are Hermida and Jacobs ... Applications to modal logic Moss ...

Generalisations to the metric setting via enriched category theory are studiend in Rutten, Worrell, ...

A.6 Solving recursive equations

Defining mathematical domains in which recursive equations can be solved is an important application of coalgebras. In fact, in the work of Barwise, the existence of unique solutions to certain equations is an axiom of non-well founded set theory and equivalent to Aczel's axiom ... AFA ... In our terms, both correspond to working with the final coalgebra of the powerset functor $\mathcal{P}$.

... tbc ...

A.7 Structural Operational Semantics

The ideas go back Rutten and Turi [44, 45, 49], but the break-through paper establishing a one-to-one correspondence betwee certain syntactic formats of structural operational semantics (such as the well-known GSOS) and distributive laws is Turi and Plotkin [48]. Further milestons on this topic include the PhD theses of Bartels [9] (see also [8, 10] and Klin [30] (see also [29]) as well as work by ... [] and []

... short summary of SOS to be added ...

... finish with the remakr that bialgebras are coalgebras over algebras

A.8 Coalgebras over algebras

At the moment there does not yet seem to be a satisfactory fully general theory of coalgebras over algebras, but there are many interesting examples. General themes are

- algebraic structure adds expressiveness to coinductive definitions
- algebraic structure adds memory to coalgebraic automata

A.8.1 Context free languages.

The example of context free languages discussed in the course is very suggestive and is due to Winter, Bonsangue and Rutten [50, 51, 52].

In terms of 'solving recursive equations', see Section 3.3, it fits into the idea of extending coinduction for T-colagebras with more expressive formats, see the remarks at the end of Section A.6. This line of thinking is closely connected to bialgebras, see Section A.7, and was one of the main topic of Bartels [9]. But we know from Beck's theorem on distributive laws, see Section **??**, that bialgebras are just another way of looking at (certain categories of) coalgebras over algebras.

A.8.2 Non-determinstic automata: coalgebras over Kleisli categories

We discussed in detail, see Section **??** and **??**, deterministic automata as coalgebras

$$X \to 2 \times X^A.$$

What, then, are non-deterministic automata? The answer is a good example of the modularity offered by category theory: If non-determinism is captured by powerset the following suggests itself.

Non-deterministic automata are coalgebras

$$X \to 2 \times (\mathcal{P}X)^A$$

But how do we capture language equivalence in this setting?

Coalgebraic behavioural equivalence as in Section **??** will not give us language equivalence, but a variation on the notion of bisimilarity as familiar from modal logic, see **??**.

The answer was given by Power and Turi [35]. First, observe that $2 \times (\mathcal{P}X)^A \cong \mathcal{P}(1+A\times X)$. Now the crucial idea of Power and Turi is to consider the coalgebra

$$X \to \mathcal{P}(1 + A \times X)$$

as a coalgebra

$$X \to 1 + A \times X$$

in the Kleisli category of the monad $\mathcal{P}$.

Power and Turi go on to show that ... They establish the principle that trace semantics arises from a "distributive law between a behaviour endofunctor for determinism (such as $1 + A \times \mathrm{Id}$) and a monad for non-determinism (such as $\mathcal{P}$)".

A.8.3 Trace semantics: Coalgebras over Kleisli categories

Hasuo, Jacobs, Sokolova, see [], generalise the ideas above to other monads than $\mathcal{P}$, in particular to the monad for probability distribution but also to all commutative monads (??)

REVISE Monads give rise to Kleisli categories and Eilenberg-Moore categories, see Section **??**. So we may also think of coalgebras over a Kleisli category. An arrow in a Kleisli category of the monad $M : \mathcal{C} \to \mathcal{C}$ (and these are the monads as in Hasekell) is an arrow

$$X \to MX$$

in $\mathcal{C}$. So in order to extend a functor $T : \mathcal{C} \to \mathcal{C}$ to the Kleisli category we need to be able to apply T to $X \to MX$ and get an arrow of type $TX \to TMX$, that is we need a distributive law

$$TMX \to MTX$$

in order to build $T(X \to MX)$ as

$$TX \to TMX \to MTX.$$

How can we use this to turn bisimulation-semantics into trace-semantics?

Let us look at a coalgebra

$$X \to \mathcal{P}(A \times X)$$

for labelled transition systems.

A.8.4 Trace semantics over Eilenberg-Moore algebras.

...

A.8.5 Vector Spaces

A.8.6 Presheaves

A.8.7 Nominal Sets

A.9 Kleene Theorems

Kleene's theorem in automata theory states that finite deterministic automata and regular expressions describe the same set of languages. The research into coalgebraic generalisations of Kleene's theorem has been initiated by Bonsangue, Rutten, Silva [12, 47].

Roughly speaking the basic idea is as follows. Kleene's theorem shows that regular expressions can be understood as a syntax to denote elements of the 'finite part' of the final coalgebra for $TX = 2 \times X^A$.

Can we define 'regular T-expressions' for general T?

Let us first think about deterministic automata again. Regular expressions are built according to

$$e ::= 0 \mid 1 \mid e + e \mid e \cdot e \mid e^*$$

$0, +$ is the semiring structure from the powerset monad. It appears here for a special reason (that can be investigated coalgebraically), namely that deterministic and non-deterministic automata accept the same class of languages.[33] This leaves $1, \cdot, ^*$. The Kleene star $*$ appears simply because coalgebras are structures that allow cycles. So let us ask now about

$$1 \qquad \text{and} \qquad e \cdot e$$

We need to find out how they are related to

$$2 \times X^A$$

Now this is not too hard to see. The regular expression 1 stands for the language that accepts the empty word, that is, the element $1 \in 2$. And as we are talking about the monoid A^* , we can replace the binary $e_1 \cdot e_2$ by unary operations ae for each $a \in A$. But these unary operations can be extracted from the part X^A of the functor.

To summarise, in the classical example of regular expressions for finite deterministic automata, there is a suggestive path from the functor $TX = 2 \times X^A$ to the syntax of regular expressions.

Can we generalise these ideas to arbitary functors $T : \mathsf{Set} \to \mathsf{Set}$?

Following on from the work cited above, this question has been answered affirmatively in the thesis of Myers [34]. Further work on coalgebraic Kleene theorems includes [46].

A.10 Modal Logic

In this section we will look at coalgebras

$$X \to \mathcal{P}X,$$

which are known in modal logic as Kripke frames, or also as Kripke models over an empty set of atomic propositions. If there is a non-empty set AtProp Kripke models coincide with coalgebras

$$X \to 2^{\mathsf{AtProp}} \times \mathcal{P}X,$$

with $X \to 2^{\mathsf{AtProp}}$ assigning to each $x \in X$ the set of atomic propositions true in that state.

[33] And, indeed, the proof of Kleene's theorem translates regular expressions to non-deterministic automata and then uses that non-deterministic automata can be made deterministic (via Kan extensions (this is just a footnote to a footnote)).

What is modal logic? The language of basic modal logic has atomic propositions, Boolean connectives and unary connectives $\Box$ and $\Diamond$ as given by the grammar

$$\phi ::= p \mid \top \mid \bot \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \Box\phi \mid \Diamond\phi$$

where p ranges over (a possibly empty) AtProp.

Definition A.4. *The semantics of a modal formula is given wrt a coalgebra* $\langle o, t\rangle : X \to 2^{\mathsf{AtProp}} \times \mathcal{P}X$ *and* $x \in X$ *via*

$$\begin{array}{ll}
x \Vdash \top & \\
x \Vdash p & o(x)(p) = 1 \\
x \Vdash \phi \wedge \psi & x \Vdash \phi \text{ and } x \Vdash \psi \\
x \Vdash \phi \vee \psi & x \Vdash \phi \text{ or } x \Vdash \psi \\
x \Vdash \neg\phi & \text{not } x \Vdash \phi \\
x \Vdash \Box\phi & y \Vdash \phi \text{ for all } y \in t(x) \\
x \Vdash \Diamond\phi & y \Vdash \phi \text{ for some } y \in t(x)
\end{array}$$

Fundamental results emphasise the importance of bisimilarity for this modal logic. For example, a theorem by van Benthem [] characterises modal logic as the fragement of first-order logic invariant under bisimilarity. Finite Kripke models are characterised up to bisimilarity by a modal formula (exercise). A similar theorem holds for general Kripke models and infinitary modal logic attributed, known as the Hennesy-Milner theorem in programm semantics (and modal logic) [] and ... see also Baltag in [].

At the beginning of all of this is the following

Exercise A.5. Show that if f is a T-coalgebra morphism for $T = 2^{\mathsf{AtProp}} \times \mathcal{P}$, then $x \Vdash \phi \;\Leftrightarrow\; f(x) \Vdash \phi$.

Remark A.6. ... this can be reformulated as: modal formulas are invariant under bisimulation ...

The following will indicate how this can be generalised from Kripke models to coalgebras. Given an arbitrary $T : \mathsf{Set} \to \mathsf{Set}$, what are the modal operators of a logic for T-coalgebras?

There are three different answers.

- Moss [] showed that we can take T itself as a modal operator, most often written ∇.
- Pattinson [] showed that $\Box$ and $\Diamond$ give rise to natural transformations, now called predicate liftings,

  $$2^{TX} \to 2^X.$$

  Conversely, any such natural transformation can be expressed by a modal formula in of 'rank 1' (that is without nested modalities) in one propositional variable.

- Bonsangue and Kurz [] showed that the modal logics for T-correspond to functors L on Boolean algebras that have a presentation by operation and equations. Moreover, for every T there is a canonical L_T given by duality. Conversely, the functors $\mathsf{BA} \to \mathsf{BA}$ that arise in this way can be characterised as those preserving so-called sifted colimits [].

The specific coalgebraic aspect of the approaches above is to work parametrically in the functor $T : \mathsf{Set} \to \mathsf{Set}$ and give uniform definitions of modal logics for each T.

This is closely related but different from earlier work in domain theory and programming languages where program logics (=modal logics) are defined for a variety of type constructors (=functors), by restricting attention to a family of functors defined inductively from a finite list of basic constructors (such as $+, \times, \to$).

A.10.1 Moss's cover modality

A.10.2 Predicate liftings

A.10.3 Presentations of functors

A.10.4 Duality theory

A.10.5 Further topics and references

A.11 Simulation, Bisimulation and other Equivalences

A.12 Coalgebras over other base categories

A.12.1 Relations

A.12.2 Preorders and Posets

A.12.3 Domains

A.12.4 Topological spaces

B Elements of the category theory of coalgebras

This appendix aims at giving a high level overview over some elements of the category theory of coalgebras. Standard category theoretic definitions not repeated in the text below can be found in Mac Lane or the nLab.

B.1 Coalgebras

Definition B.1. *Given a category $\mathcal{X}$ and a functor $T : \mathcal{X} \to \mathcal{X}$, a **coalgebra** (X, ξ) is consists of an object X in $\mathcal{X}$ an arrow $\xi : X \to TX$ in $\mathcal{X}$. A coalgebra homomorphism from a coalgebra (X, ξ) to a coalgebra $(X', \xi)'$ is an arrow $f : X \to X'$ in $\mathcal{X}$ such that $\xi \circ f = Tf \circ \xi$ or, in a diagram,*

$$\begin{array}{ccc} X & \xrightarrow{\xi} & TX \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle Tf} \\ X' & \xrightarrow[\xi]{} & TX' \end{array}$$

Given a coalgebra (X, ξ) one often calls X the *underlying set* and ξ the *structure* of the coalgebra.

If the category $\mathcal{X}$ we have in mind is sufficiently similar to the category Set of sets and functions, we also call X the state space and elements of X states and ξ the transition function. There are many important examples. To fix ideas, I just give two for now.

Example B.2. Let $\mathcal{X} = \mathsf{Set}$ be the category of sets and functions.

1. Let A be a fixed set and $TX = A \times X$. Then we can think of coalgebras as dynamic systems that output an element of A at every step.
2. Let $\mathcal{P}X = \{Y \mid Y \subseteq X\}$ be the so-called powerset functor. Then we can think of a coalgebra $X \to \mathcal{P}X$ as a relation on X.

The next exercise is crucial. If we think of a coalgebra (X, ξ) as a dynamic system and of an element $x \in X$ as the initial state of a process (X, ξ, x) then a coalgebra morphism $f : (X, \xi) \to (X', \xi)$ maps a process x to a process $x' = f(x)$ with the same behaviour. The purpose of the next exercise is to work out this notion of behaviour for two different examples of T. The first should be relatively straight forward. The second one is more delicate and its discovery by Aczel in 1989 can be seen as an early starting point of the theory of coalgebras.

Exercise B.3.
1. Let $TX = A \times X$ and let $f : (X, \xi) \to (X', \xi')$. Describe explicitely what it means that $x' = f(x)$.
2. Let $\mathcal{P}X$ be the so-called powerset functor. We have to say what $\mathcal{P}$ does on maps. For $f : X \to Y$ define $\mathcal{P}f : \mathcal{P}X \to \mathcal{P}Y$ as direct image. Describe explicitely what it means that $x' = f(x)$.

Now where we have some intuitive understanding of what the notion of coalgebra morphism captures, the next step is to check that coalgebras and coalgebra morphisms organise themselves in a category, which allows us to make

Definition B.4. *Given a category $\mathcal{X}$ and a functor $T : \mathcal{X} \to \mathcal{X}$, write* $\mathsf{Coalg}(T)$ *for the category that has coalgebras as objects and coalgebra homomorphisms as arrows.*

B.2 Some structure theorems

Some of the basic theory of coalgebras consists of theorems that establish properties of $\mathsf{Coalg}(T)$ given properties of T. Actually, rather than working with the category $\mathsf{Coalg}(T)$, it is more approriate to formulate properties in terms of the so-called forgetful[34] functor $U : \mathsf{Coalg}(T) \to \mathcal{X}$ which maps arrows $f : (X, \xi) \to (X', \xi')$ to arrows $f : X \to X'$. Depending on your background in category theory, you may take the following two exercises as a reference of useful results to come back to later or you may attempt some.[35]

Exercise B.5. Let $\mathcal{X}$ be a category and a functor $T : \mathcal{X} \to \mathcal{X}$.

1. If $\mathcal{X}$ has an initial object, then $\mathsf{Coalg}(T)$ has an initial object.
2. If Uf is an epi in $\mathcal{X}$, then f is an epi in $\mathsf{Coalg}(T)$.
3. If Uf is a mono in $\mathcal{X}$ and T preserves monos, then f is a strong mono in $\mathsf{Coalg}(T)$.
4. If $\mathcal{X}$ has a type of colimit, then $\mathsf{Coalg}(T)$ has this type of colimit and U creates it.
5. If $\mathcal{X}$ has a type of limit and T preserves it, then $\mathsf{Coalg}(T)$ has this type of colimit and U creates it.
6. If $(\mathcal{E}, \mathcal{M})$ is a factorisation system of $\mathcal{X}$ and T preserves monos, then $(U^{-1}\mathcal{E}, U^{-1}\mathcal{M})$ is a factorisation system for $\mathsf{Coalg}(T)$.

The category $\mathcal{X} = \mathsf{Set}$ is of special interest. Many of the properties below can also be established for categories similar to Set such as various categories of topological spaces.

Exercise B.6. Let $T : \mathsf{Set} \to \mathsf{Set}$ be a functor.

1. $\mathsf{Coalg}(T)$ has all colimits and they are computed as in Set.
2. Unions of arbitrary (possibly large) families of subcoalgebras exist and are computed as in Set.

[34] The reason for the terminology 'forgetful' should be obvious, but also can be given a technical interpretation, namely to mean that U is faithful.

[35] Definitions can be found in n-lab. Proofs are routine for category theorists, but could be a good exercise for somebody who wants to learn the corresponding category theoretic notions. Most of the proofs were spelled out in the appendix to my PhD thesis.

3. $\mathsf{Coalg}(T)$ has equalisers and they are computed as unions of subcoalgebras.

4. Epis and strong monos form a factorisation system in $\mathsf{Coalg}(T)$.

B.3 Bisimulation and Behaviour

Having seen the defintion of $\mathsf{Coalg}(T)$ and some of its basic structure, we now formalise the notion of behaviour implicit already in Exercise B.3. For this we assume that $\mathsf{Coalg}(T)$ has a forgetful functor to Set, even though generalisations are possible.

In the following we use the notation $\mathbb{X}$ to suggest that $\mathbb{X} = (X, \xi)$ is a coalgebra, but the definition does make sense also for other categories.

A coalgebra $\mathbb{X} = (X, \xi)$ together with a state $x \in X$ can be thought of as a *process* $(\mathbb{X}, x)$ starting to compute in the initial state x with computation steps given by ξ.

Definition B.7. *Let $U : \mathcal{C} \to \mathsf{Set}$ and $\mathbb{X}, \mathbb{X}' \in \mathcal{C}$. **Behavioural equivalence** is the smallest equivalence relation $\simeq$ generated by all pairs*

$$(\mathbb{X}, x) \simeq (\mathbb{X}', x')$$

such that $\mathbb{X}, \mathbb{X}'$ are objects of $\mathcal{C}$ and $x \in U\mathbb{X}$ and $x' \in U\mathbb{X}'$ and there are morphisms $f : \mathbb{X} \to \mathbb{X}''$, $f' : \mathbb{X}' \to \mathbb{X}''$ with

$$f(x) = f'(x').$$

We may abbreviate $(\mathbb{X}, x) \simeq (\mathbb{X}', x')$ to $x \simeq x'$.

The definition formalises the idea that

behaviour is what remains invariant
under the morphisms of a category.

Exercise B.8. Show that the notion of behavioural equivalence coincides in Example B.2 with the well-known notion of bisimulation.

For the category theorist among the readers the following will be of interest.

Exercise B.9. Two processes are behaviourally equivalent iff they are in the same connected component of the category of elements of U.

B.4 Final coalgebras

It is often possible to find one coalgebra that is universal in the sense that is consists of all possible behaviours. Because of its importance to coalgebra we recall

Definition B.10. *Let $\mathcal{C}$ be a category. An object C in $\mathcal{C}$ is called **final** or **terminal** if for all objects A in $\mathcal{C}$ there is a unique arrow $A \to C$.*

The next exercise shows how the data type of streams arises as a final coalgebra.

Exercise B.11. Let $T : \mathsf{Set} \to \mathsf{Set}$ be the functor $TX = A \times X$ for some set A. Show that the terminal coalgebra has as carrier the set $A^{\mathbb{N}}$ of infinite lists or streams over A. What is the structure $A^{\mathbb{N}} \to T(A^{\mathbb{N}})$ of the final coalgebra?

Similarly, we have

Exercise B.12. Let $T : \mathsf{Set} \to \mathsf{Set}$ be the functor $TX = 1 + A \times X$. Show that the final coalgebra has as carrier the set of finite and infinite lists. What is the structure of the terminal coalgebra?

For the student who wants to learn more about category theory the following exercise will be instructive.

Exercise B.13. Show that the carrier of the final coalgebra is given by the colimit of $U : \mathsf{Coalg}(T) \to \mathsf{Set}$ and that the structure is given by its universal property.

The significance of this observation is the following. It is well-known that an element of the colimit of U is exactly a connected component of its category of elements, that is, an equivalence class of behaviourally equivalent elements. We conclude that the final coalgebra, if it exists, is a coalgebra which has as its carrier exactly the equivalence classes of behaviourally equivalent processes.

An important property of a final coalgebra $X \to TX$ is that it is, like any limit, determined up to unique isomorphism, which explains why we often speak of 'the' final coalgebra.

Coalgebras as generalised postfix points. Every monotone operation on a complete lattice has a smallest and largest fixpoint (Knaster-Tarski). This can be seen as a special case of our setting. A complete lattice, as any poset or preorder, is a particularly simple example of a category $\mathcal{X}$ and to say that T is a monotone operation on $\mathcal{X}$ is exactly to say that $T : \mathcal{X} \to \mathcal{X}$ is a functor (exercise!). A coalgebra $X \to TX$ then is exactly a postfix point of T, as $X \to TX$ is now the same as $X \leq TX$.

Exercise B.14. Show that the largest postfix point is a fix point.

The exercise above is very instructive if done in conjunction with the next exercise. It shows a general principle, namely that many category theoretic constructions can be understood as generalisations of much simpler lattice theoretic ones.

Exercise B.15 (Lambek's lemma)**.** Let $\mathcal{X}$ be any category and $T : \mathcal{X} \to \mathcal{X}$ be any functor. If (X, ξ) is a final coalgebra, then ξ is an isomorphism.

The proof of Lambek's lemma is a direct generalisation of the proof of the corresponding lattice theoretic fact. It is also worth observing that it uses all of the axioms of a category and a functor. So we may say that category theory axiomatises exactly what is needed to get Lambek's lemma.

Existence of final coalgebras. When does a final coalgebra exist? Roughly speaking, if the base category $\mathcal{X}$ has enough limits and if the functor T has some 'size restriction'. For example, the functor $TX = A \times X$ is determined on arbitrary sets by what T does on finite sets. So the final coalgebra does exists. Technically, this is a consequence of the adjoint functor theorem (exercise!). The example $TX = \mathcal{P}X$ is different. We cannot have a final coalgebra because there cannot be an isomorphism $X \to \mathcal{P}X$ in Set (Cantor's theorem).

On the other hand, if we extend $\mathcal{P}$ from sets to classes by $\mathcal{P}X = \{\mathcal{P}Y \mid Y \subseteq X, Y \text{ a set } \}$, then the final coalgebra does exists and, as shown by Aczel and Mendler, this works for any functor on Set.

B.5 Duality

In category theory, the word concrete often refers to a forgeful functor to Set: A **concrete category** is a faithful functor $\mathcal{C} \to \mathsf{Set}$. Abstract duality is purely formal, but concrete dualities are very interesting: Given $\mathcal{C} \to \mathsf{Set}$, can we describe a functor $\mathcal{C}^{\mathrm{op}} \to \mathsf{Set}$?

B.5.1 Abstract duality

For every category $\mathcal{C}$ there is the dual category $\mathcal{C}^{\mathrm{op}}$. For example, the operation

$$2^{-} : \mathsf{Set} \to \mathsf{Set}$$

which maps a function $f : X \to Y$ to the function $2^f : 2^Y \to 2^X$ (which maps a function $b : Y \to 2$ to $b \circ f : X \to 2$) is not a functor as it does not preserves but "turns around" arrows. We can either call it a *contravariant functor* or we can call it a functor

$$2^{-} : \mathsf{Set}^{\mathrm{op}} \to \mathsf{Set}$$

or, equivalently, a functor

$$2^{-} : \mathsf{Set} \to \mathsf{Set}^{\mathrm{op}}.$$

The operation of turning around arrows is a purley formal one. As in the example above, its main obvious value is one of simplification. We have seen that it allows to reduce the notion of a contravariant functor to that of a functor. Similarly, a colimit in $\mathcal{C}$ is a limit in $\mathcal{C}^{\mathrm{op}}$. More generally every definition and every theorem in category theory has a dual definition and theorem.

For example, we may define an algebra as a co-coalgebra. More precisely, given $T : \mathcal{C} \to \mathcal{C}$, we may define $\mathsf{Alg}(T) = (\mathsf{Coalg}(T^{\mathrm{op}}))^{\mathrm{op}}$, where T^{op} is the same as T considered as a functor $\mathcal{C}^{\mathrm{op}} \to \mathcal{C}^{\mathrm{op}}$. We call an object of $\mathsf{Alg}(T)$ an algebra for the functor T.

Example B.16. A monoid is an algebra for the functor $TX = 1 + X \times X$.

Note that being an algebra for $TX = 1 + X \times X$ does not enforce any equations between terms.

Exercise B.17. 1. Check that in the example above, an arrow in $\mathsf{Alg}(T)$ coincides with the usual notion of monoid homomorphism.

2. Consider algebras of any given type Ω (see wikipedia on universal algebra). Define a functor T so that $\mathsf{Alg}(T)$ coincides with (is ismoprhic to) the category of algebras of type Ω.

The exercise shows that all the common algebras such as monoids, groups, rings, lattices, etc are also algebras for a functor $\mathsf{Set} \to \mathsf{Set}$.

We can now use duality to prove that the structure of an initial algebra is an isomorphism. First, we recall that an initial object in $\mathcal{C}$ is a final object in $\mathcal{C}^{\mathrm{op}}$.

Exercise B.18. Reformulate the notion of an initial algebra without making use of $(-)^{\mathrm{op}}$.

Now we could either prove the following as an exercise from scratch, or we could say that it is the dual of Exercise B.15.

Exercise B.19. The structure of an initial algebra is an isomorphism.

The abstract duality between algebras and coalgebras can be used to transfer some interesting theorems known from universal algebra to coalgebra.

One technical complication that arises is that there is no category theoretic axiomatisation of injective and surjective maps. Often one can take the category theoretic notion of a mono to axiomatize the set-theoretic notion of an injective map and the notion of an epi to axiomatize surjection. But this is only an approximation. For example, there are categories of algebras in which epis do not need to be surjective. The solution is to use the category theoretic notion of a factorisation system and to parameterize the theory of algebra/coalgebras with a factorisation system. The advantage of doing this is that the dual of a factorisation system is a factorisation system (and this way one can make it happen that injections dualise to surjections and vice versa).

Example of theorems of universal algebra that dualise that way are the isomorphism theorems and Birkhoff's variety theorem.

On the other hand, many of the deeper notions do not dualize. One way of explaining this is that $\mathsf{Alg}(T) = (\mathsf{Coalg}(T^{\mathrm{op}}))^{\mathrm{op}}$ says that algebras over Set are dual to coalgebras over $\mathsf{Set}^{\mathrm{op}}$ and Set and $\mathsf{Set}^{\mathrm{op}}$ are very different categories. The easy category theoretic duality can only be applied for category theoretic properties that are shared by Set and $\mathsf{Set}^{\mathrm{op}}$.

B.5.2 Concrete dualities

The duality of $\mathsf{Set}^{\mathrm{op}}$ and Set is trivial. But what is interesting is the question of how to represent the "turned around" arrows in $\mathsf{Set}^{\mathrm{op}}$ as functions. In other

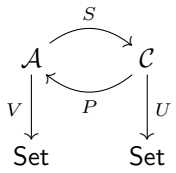
words, is there a category equivalent to $\mathsf{Set}^{\mathrm{op}}$ which has a forgetful functor to Set?

Exercise B.20. This exercise requires some knowledge of Boolean algberas and quite a bit of time. But one can get the gist by reading through the items below. Let Set_{fin} be the category of finite sets. Show $2^- : \mathsf{Set}_{fin}^{\mathrm{op}}$ is equivalent to the category of finite Boolean algebras BA_{fin}.

1. For every X, the set 2^X can be equipped with the structure of a Boolean algebra (inherited pointwise from the usual Booelan structure of the truth values in $2 = \{0, 1\}$).
2. For every function $f : X \to Y$, we have that $2^f : 2^Y \to 2^X$ is a Boolean algebra homomorphism.
3. The two previous item can be summarised by saying that the functor $2^- : \mathsf{Set}_{fin}^{\mathrm{op}} \to \mathsf{Set}_{fin}$ restricts to a functor $2^- : \mathsf{Set}_{fin}^{\mathrm{op}} \to \mathsf{BA}_{fin}$.
4. Every finite $A \in \mathsf{BA}$ is isomorphic to an algebra of the form 2^X (Hint: Take X to the set of atoms of A).
5. Every Boolean algebra homomorphism $2^Y \to 2^X$ between finite Boolean algebras is of the form 2^f for some $f : X \to Y$.
6. $2^- : \mathsf{Set}_{fin}^{\mathrm{op}} \to \mathsf{BA}_{fin}$ is an equivalence of categories.

This dual equivalence can be extended to a dual adjunction between Set and BA and to a dual equivalence between so-called Stone spaces and BA. The standard reference to this subject is Johnstone's book.

A situation encountered often is



(54)

where P, S are a dual adjunction and U and V have left adjoints, F and G, respectively. Then one can show that the dual adjunction arises from 'homming into a dualising object' that simultaneously lives in $\mathcal{A}$ and $\mathcal{C}$.

Exercise B.21. Consider the situation of the diagram above. Recall that the adjointness assmuptions amount to $\mathcal{A}(A, PC) \cong \mathcal{C}(C, SA)$ and $\mathcal{A}(GY, A) \cong \mathsf{Set}(Y, VA)$ and $\mathcal{C}(FY, C) \cong \mathsf{Set}(Y, UC)$. (Hint: Use $Y \cong \mathsf{Set}(1, Y)$.)

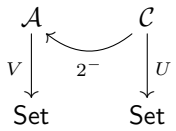
1. Show that $UPA \cong \mathcal{A}(-, SF1)$ and $VSX = \mathcal{C}(-, PG1)$. In other words, P and S arise from homming into $SF1$ and $PG1$, respecitvely.

2. $VSF1 = UPG1$. In other words, $SF1$ and $PG1$ can be considered to be the same dualising object, equipped with an $\mathcal{A}$-structure and an $\mathcal{C}$-structure.

From a coalgebraic point of view, thinking of $\mathcal{C}$ as a category of coalgebras, it opens the possibility to not only formally dualise coalgebras to algebras, but to represent these dual algebras concretely as algebras over Set. We can then think of the algebras as logics for coalgebras: For any coalgebra C, the algebra PC is the theory of C, or the algebra of predicates or propositions of C. We will look at an example in the next section.

B.6 Algebras as logics for coalgebras

We continue from the last section and specialise (54) to



Assume again that G is a left adjoint of V. Let $X \in \mathcal{C}$. Then there is a unique homomorphism from the initial algebra $G0$ to X,

$$[\![-]\!] : G0 \to 2^X,$$

and we can define for all $x \in UX$ and $a \in G0$

$$x \Vdash a \;\Leftrightarrow\; x \in [\![a]\!].$$

Example B.22. The paradigmatic example that links logics for coalgebras with modal logic arises from $TX = \mathcal{P}X$. In this case coalgebras are what is known as Kripke frames in modal logic. The functor 2^- takes a coalgebra $\xi : X \to TX$ and maps it to an algebra

$$2^X \xrightarrow{\Box} 2^{\mathcal{P}X} \xrightarrow{2^\xi} 2^X$$

where 2^X is the Boolean algebra of subsets of X and $\Box$ is defined in such a way that $2^\xi \circ \Box$ gives the semantics of the modal Box operator, see the next exercise.

Exercise B.23. Define $\Box$ so that $x \in \Box a \;\Leftrightarrow\; \forall y \in \xi(x) \,.\, y \in a$.

As hinted at in this example, one can set up things in such a way that modal logic as we love and know it appears as being dual to the category of $\mathcal{P}$-coalgebras. Formalizing this idea in the language of category theory allows us to do this parametrically in T, so that we obtain a general account of logics for T-coalgebras parameterised in the type T.

Without going any further, let us say that this can be taken as the beginning of a general theory of logics for coalgebras which centres around the following questions.

- Given a functor $T : \mathsf{Set} \to \mathsf{Set}$, what is a logic for T-coalgebras?
- Depending on T what are properties of the logic we can guarantee (invariance under behavioural equivalence, soundness, completeness, expressiveness)?
- How to design logics that capture various notions of behavioural equivalence/bisimulation (trace equivalence, various simulations, ...)?
- How to extend basic coalgebraic logic in a systematic way (mu-calculus, hybrid logic, ...)
- What are good proof systems for coalgebraic logics?
- (tobecompleted)

B.7 Natural Transformations and the Yoneda Lemma

Functors are structure preserving maps between categories and natural transformations are structure preserving maps between functors. Thus, it would have been natural, indeed, to introduce natural transformations right from the start. In fact, the first paper on category theory was written in order to formalize the intuitive idea of a natural transformation. The reason to put natural transformations here is that on the one hand we have now enough examples to illustrate the power and importance of that notion and on the other hand they will become centre stage in the following sections.

tbwritten

B.8 Monads

The notion of a monad is a category theoretic axiomatisation of terms modulo equations. For example, if M is the monad of abelian groups, then MX is the set of terms one can form using the constant 0 and the binary operation $+$ and the variables $x \in X$, modulo the equations of an abelian group. For example, if $X = \{x, y\}$, then $x + y$ and $y + x$ denote the same element of MX. In analogy with monoids, η is called the *unit* and μ is called the *multiplication* of the monad.

Formally, a short way to say what a monad is "a monoid in the category of endofunctors". Spelling this out, a monad (M, η, μ) on a category $\mathcal{C}$ is a functor $M : \mathcal{C} \to \mathcal{C}$ and natural transformations

$$\eta : 1 \to M \qquad \mu : MM \to M$$

(where 1 here denotes the identity functor on $\mathcal{C}$) such that $\mathrm{id} = \mu \circ M\eta = \mu \circ \eta M$ and $\mu \circ \mu M = \mu \circ M\mu$.

The first example is an example where terms and operations are lists. We show in detail how this monad corresponds to an algebraic theory given by operations and equations as familiar from universal algebra.

Example B.24. Let $MX = X^*$ be the set of finite words over X and $\eta(x) = x$ and $\mu(w_1 \dots w_n) = w_1 \dots w_n$, where $w_1 \dots w_n$ on the left denotes a word of words and on the right denotes the word obtained from concatenation the w_i. This monad is sometimes called the list monad, since we can think of a word $x_1 \dots x_n$ as a list $[x_1, \dots x_n]$. If we want to bring this in more familiar algebraic notation, then we choose a set

$$\Sigma = \{[]_n \mid n \in \mathbb{N}\}$$

of operation symbols with the operation symbol $[]_n$ denoting an operation of arity n and write $[]_n(x_1, \dots, x_n)$ instead of $[x_1, \dots x_n]$. The multiplication of the monad takes a list of lists and flattens it to a list using concatenation. The equation $\mu \circ M\eta = \mathrm{id}$ can be written explicitely as a family of equations indexed by $n \in \mathbb{N}$

$$[[x_1], \dots [x_n]] = [x_1, \dots x_n]$$

or, in algebraic notation,

$$[]_n([]_1(x_1), \dots []_1(x_n)) = []_n(x_1, \dots x_n)$$

and the equation $\mu \circ \eta M = \mathrm{id}$ as

$$[[x_1, \dots x_n]] = [x_1, \dots x_n]$$

which is

$$[]_1([]_n(x_1, \dots x_n)) = []_n(x_1, \dots x_n)$$

in algebraic notation. The equation $\mu \circ \mu M = \mu \circ M\mu$ corresponds to a family of equations, one for each m-tuple of numbers $n_1, \dots n_m$,

$$[[x_{1,1}, \dots x_{1,n_1}], \dots [x_{m,1}, \dots x_{m,n_m}]] = [x_{1,1}, \dots x_{1,n_1}, \dots x_{m,1}, \dots x_{m,n_m}]$$

which is

$$\begin{aligned} []_m([]_{n_1}(x_{1,1}, \dots x_{1,n_1}), \dots []_{n_m}(x_{m,1}, \dots x_{m,n_m})) = \\ = []_{n_1+\dots n_m}(x_{1,1}, \dots x_{1,n_1}, \dots x_{m,1}, \dots x_{m,n_m}) \end{aligned}$$

in algebraic notation.

This example suggests that the list monad describes the algebraic theory of monoids. This can be made precise by introducing the notion of algebra for a monad.

Definition B.25. *An algebra for a monad (M, η, μ), also called an Eilenberg-Moore algebra, is an arrow $\alpha : MX \to X$ such that $\alpha \circ \eta X = \mathrm{Id}$ and $\alpha \circ \mu X = \alpha \circ M\alpha$. The category M-Alg of algebras for the monad M is the corresponding full subcategory of algebras for the functor M.*

Intuitively, the terms corresponding to algebras for the functor M are trees where each node conists of a list of children. Algebras for the monad M have the additional operation of flattening lists of lists, so that terms are just lists.

Fact B.26. *Given $MX = X^*$, the category M-Alg is isomorphic to the category of monoids.*

The above shows that the same algebraic theory can be represented in different ways. From the monad point of view, we see for each $n \in \mathbb{N}$ an operation that forms a list of length n and equations that flatten lists. From the more familiar monoid point of view, we can represent the same theory by a constant 0 and a binary operation + plus equations for associativity.

In the second example, the multiplication is doing more than just substituting lists into lists. It also eliminates duplicates and order.

Example B.27. The (finite or not) powerset functor is a monad, with $\eta(x) = \{x\}$ and $\mu(\mathcal{S}) = \bigcup \mathcal{S}$.

To make the equations in the monad $\mathcal{P}_{fin}$ more visible note that there is a natural transformation

$$e : X^* \to \mathcal{P}_{fin} \tag{55}$$

which identifies two lists if and only if they are the same modulo the equations specifying that the order of the list doesnt matter and that one can eliminate duplicates from lists.

The last example suggest that the monad $\mathcal{P}_{fin}$ is the theory of semi-lattices, that is, the theory given by a constant 0 and a binary operation $\vee$ satisfying the laws of an idempotent, commutative monoid, explicitely,

$$\begin{aligned} 0 \vee y &= y \\ x \vee 0 &= x \\ (x \vee y) \vee z &= x \vee (y \vee z) \\ x \vee y &= y \vee x \\ x \vee x &= x \end{aligned}$$

Fact B.28. *Let M be the monad $\mathcal{P}_{fin}$ described above. Then the category M-Alg is isomorphic to the category of semi-lattices. If $M = \mathcal{P}$ without the finiteness restriction then M-Alg is the category of complete semi-lattices.*

That the notion of a monad M captures the inution of MX as the terms over X is supported by the following

Exercise B.29. Make the following precise. Let F be left-adjoint to U. Then UF is a monad. Conversely, if M is a monad, then there is an adjunction $F \dashv U$ such that $UF = M$.

In the exercise above we think of the left adjoint F as constructing free algebras. At least in the case $\mathcal{C} = \mathsf{Set}$ this is justified by the following.

Theorem B.30. *Let M be monad on Set. Then there is a class of operations Σ and a class of equations E such that M-Alg is isomorphic to the category of algebras given by (Σ, E). (Conversely, if $\mathcal{A}$ is the class of algebras for operations and equations and the forgetful functor $\mathcal{A} \to \mathsf{Set}$ has a left adjoint, then $\mathcal{A}$ is ismorphic to M-Alg for some $M : \mathsf{Set} \to \mathsf{Set}$.)*

The importance of this theorem to us is that it justifies to replace the syntactic notion of terms given by operations and equations by the abstract and axiomatic notion of a monad. This greatly simplifies abstract reasoning. In addition it allows us to generalise to monads on other categories than Set.

B.9 Distributive laws and bialgebras

In these notes the interaction between algebraic and coalgebraic structure plays an important role. Mathematically, this can be captured by the notion of a so-called distributive law. A distributive law always involves two functors $M, T : \mathcal{X} \to \mathcal{X}$ and is given by a natural transformation

$$\lambda : MT \to TM$$

If M and T are mere functors than this all we can say. But already such a simple λ maybe useful. It allows us to lift M to $\mathsf{Coalg}(T)$ and to lift T to $\mathsf{Alg}(M)$ so that we can consider algebras with coalgebraic structure and coalgebras with algebraic structure. Moreover, both ways of looking at (co)algebraic structure is equivalent to a bialgebraic view.

Definition B.31. *Let $M, T : \mathcal{X} \to \mathcal{X}$ be functors and $\lambda : MT \to TM$ a natural transformation.*

1. *The category $\mathsf{Bialg}(M, T)$ has objects $MX \to X \to TX$ such that*

$$\begin{array}{ccccc} MX & \longrightarrow & X & \longrightarrow & TX \\ & \searrow & & & \nearrow \\ & & MTX & \xrightarrow[\lambda]{} & TMX \end{array} \tag{56}$$

   commutes. Morphisms are arrows that are both M-algebra and T-coalgebra morphisms.

2. *$\tilde{M} : \mathsf{Coalg}(T) \to \mathsf{Coalg}(T)$ is the functor given by $\tilde{M}(X \to TX) = MX \to MTX \to TMX$.*

3. *$\tilde{T} : \mathsf{Alg}(M) \to \mathsf{Alg}(M)$ is the functor given by $\tilde{T}(MX \to X) = MTX \to TMX \to TX$.*

The following exercise is based on the simple observation that in (56) we have that $MX \to X$ is a coalgebra morphism and $X \to TX$ is an algebra morphism.

Exercise B.32. Let $M, T : \mathcal{X} \to \mathcal{X}$ be functors and $\lambda : MT \to TM$ be a natural transformation.

1. $\tilde{M}$ lifts M, that is,

$$\begin{array}{ccc} \mathsf{Coalg}(T) & \xrightarrow{\tilde{M}} & \mathsf{Coalg}(T) \\ \downarrow & & \downarrow \\ \mathcal{X} & \xrightarrow{M} & \mathcal{X} \end{array}$$

   commutes.

2. $\tilde{T}$ lifts T, that is,

$$\begin{array}{ccc} \mathsf{Alg}(M) & \xrightarrow{\tilde{T}} & \mathsf{Alg}(M) \\ \downarrow & & \downarrow \\ \mathcal{X} & \xrightarrow{T} & \mathcal{X} \end{array}$$

commutes.

3. $\mathsf{Coalg}(\tilde{T}) \cong \mathsf{Alg}(\tilde{M}) \cong \mathsf{Bialg}(M, T)$.

An important consequence of this setup is that initial algebra semantics and final coalgebra semantics agree:

The final T-coalgebra $Z \to TZ$ is also the final bialgebra $MZ \to Z \to TZ$. The initial M-algebra $MI \to I$ is also the initial bialgebra $MI \to I \to TI$. We can define $I \to Z$ both via initiality and via finality. Both definitions coincide. So we can say that in this framework

initial algebra and final coalgebra semantics coincide.

Another important consequence is that

coalgebraic behavioural equivalence is a congruence.

Exercise B.33. Think about the two slogans in bold face above.

Moreover, if M is a monad then we would like to have that $\tilde{M}$ is a monad on $\mathsf{Coalg}(T)$ and that $\tilde{T}$ maps algebras for a monad to algebras for a monad.

Exercise B.34. Assume that (M, η, μ) is a monad. Find equations on λ, η, μ that suffice to guarantee that $\tilde{T}$ maps Eilenberg-Moore algebras to Eilenberg-Moore algebras. Which equations guarantee that $\tilde{M}$ is a monad? Which equations allows us to lift T to the Kleisli category of M?

As a side remark, the additional structure of M being a monad, allows us to recover the distributive law from knowing that T has a lifting to M-Alg:

Exercise B.35. Assume that there is a functor T' such that

$$\begin{array}{ccc} M\text{-}\mathsf{Alg} & \xrightarrow{T'} & M\text{-}\mathsf{Alg} \\ \downarrow & & \downarrow \\ \mathcal{X} & \xrightarrow{T} & \mathcal{X} \end{array}$$

Then there is a distributive law $MT \to MT$ and T' is the lifting of T induced by $MT \to MT$. (Hint: Recall that $M = UF$ for an adjunction with unit $\mathrm{Id} \to UF$ and counit $FU \to \mathrm{Id}$. Now fill in $MT \to MTM = UFTUF = UFUT'F \to UT'F = TUF = TM$.)

The original reference for results like the one above is Beck's "Distributive Laws" (1969). Beck's result concern distributive laws between two monads:

Example B.36. The first item is the original one by Beck, the second item is the variation that plays a part in the first part of the course.

1. Let A be the monad of abelian groups and let M be the monad of monoids. Show that the well-known law of rings stating that multiplication distributes over gives rise to a distributive law $MA \to AM$. This observation has several consequences: The free ring over X is AMX. Rings are algebras for the lifting of the monad A to monoids.

2. As in the previous subsection on monads, let $\mathcal{P}_{fin}$ be the monad of semilattices and let $(-)^*$ be the monad of monoids. Show that $\mathcal{P}_{fin}X^*$ is the free idempotent semiring over X.

The duality between the category of Eilenberg-Moore algebras and the Kleisli-category as well as the duality between algebras and coalgebras is best explained using Street's "Formal Theory of Monads" (1972).

B.10 Representations of Functors

The definition of an algebra or coalgebra for a functor looks very general. What is a functor? Can we describe functors explicitely in terms of more familiar operations?

This is possible at least for a large class of Set-functors called the finitary functors. The usual definition is that these are the functors that preserve so-called filtered colimits, but the following is equivalent in the special case of Set-functors.

Definition B.37. *A functor $T : \mathsf{Set} \to \mathsf{Set}$ if **finitary** iff there is a family $(A_n)_{n\in\mathbb{N}}$ of sets and a componentwise surjective transformation*

$$\tau_X : \coprod A_n \times X^n \to TX$$

natural in X. We call τ a presentation of the functor by operations and equations where the operation symbols are the elements of A_n and the equations are all pairs identified by τ.

Example B.38. (55) shows that $\mathcal{P}_\omega$ is a finitary functor. It is presented by operations $[]_n$ for each $n \in \mathbb{N}$ and equations (recall the notation from ...) such as

$$[z, x, y] = [z, y, x]$$
$$[z, x, x] = [z, x]$$

(These equations are essentially those given by the structural rules of the sequent calculus for classical logic.)

Exercise B.39. ... canonical representation ...

References

[1] S. Abramsky. Domain theory in logical form. *Ann. Pure Appl. Logic*, 51, 1991.

[2] S. Abramsky and A. Jung. Domain theory. In *Handbook of Logic in Computer Science*. OUP, 1994.

[3] P. Aczel. *Non-Well-Founded Sets*. CSLI, Stanford, 1988.

[4] P. Aczel and N. P. Mendler. A final coalgebra theorem. In *Category Theory and Computer Science*, volume 389 of *LNCS*, 1989.

[5] J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories*. CUP, 1994.

[6] S. Awodey and J. Hughes. The coalgebraic dual of Birkhoff's variety theorem. Technical Report CMU-PHIL-109, Carnegie Mellon University, Pittsburgh, PA, 15213, November 2000.

[7] M. Barr. Terminal coalgebras in well-founded set theory. *Theoret. Comput. Sci.*, 114(2), June 1993.

[8] F. Bartels. Generalised coinduction. *Math. Structures Comput. Sci.*, 13, 2003.

[9] F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

[10] F. Bartels, A. Sokolova, and E. de Vink. A hierarchy of probabilistic system types. *Theoret. Comput. Sci.*, 327, 2004.

[11] M. M. Bonsangue, H. H. Hansen, A. Kurz, and J. Rot. Presenting distributive laws. *Logical Methods in Computer Science*, 11(3), 2015.

[12] M. M. Bonsangue, J. J. M. M. Rutten, and A. Silva. A kleene theorem for polynomial coalgebras. In *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pages 122–136, 2009.

[13] C. Cîrstea and D. Pattinson. Modular construction of modal logics. In *CONCUR'04*.

[14] C. Cîrstea and D. Pattinson. Modular proof systems for coalgebraic logics. *Theoret. Comp. Sci.*, 338, 2007.

[15] E. de Vink and J. Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*, 1997.

[16] E. de Vink and J. Rutten. Bisimulation for probabilistic transition systems: A coalgebraic approach. *Theor. Comp. Sci.*, 221, 1999.

[17] D. B. Francis Borceux. *Mal'cev, Protomodular, Homological and Semi-Abelian Categorie*. 2004.

[18] R. Goldblatt. What is the coalgebraic analogue of Birkhoff's variety theorem? *Theor. Comp. Sci.*, 266, 2001.

[19] H. P. Gumm. Equational and implicational classes of colgebras. *Theor. Comp. Sci.*, 260, 2001.

[20] H. P. Gumm and T. Schröder. Types and coalgebraic structure. *Algebra Universalis*. to appear.

[21] H. P. Gumm and T. Schröder. Covarieties and complete covarieties. In *CMCS'98*, volume 11 of *ENTCS*, 1998.

[22] H. P. Gumm and T. Schröder. Coalgebraic structure from weak limit preserving functors. In *CMCS'00*, volume 33 of *ENTCS*, 2000.

[23] H. P. Gumm and T. Schröder. Coalgebras of bounded type. Technical Report 25, FG Informatik, Philipps-Universität Marburg, 2000.

[24] H. P. Gumm and T. Schröder. Covarieties and complete covarieties. *Theor. Comp. Sci.*, 260, 2001.

[25] H. P. Gumm and T. Schröder. Products of coalgebras. *Algebra Universalis*, 46(1–2), 2001.

[26] H. H. Hansen, C. Kupke, and J. Rutten. Stream differential equations: Specification formats and solution methods. *Logical Methods in Computer Science*, 13(1), 2017.

[27] I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3, 2007.

[28] B. Jacobs. Towards a duality result in the modal logic of coalgebras. `http://www.cs.kun.nl/~bart/PAPERS/duality.ps.Z`.

[29] B. Klin. The least fibred lifting and the expressivity of coalgebraic modal logic. In *CALCO'05*.

[30] B. Klin. *An Abstract Coalgebraic Approach to Process Equivalence for Well-Behaved Operational Semantics*. PhD thesis, University of Aarhus, 2004.

[31] A. Kurz. *Logics for Coalgebras and Applications to Computer Science*. PhD thesis, LMU, 2000.

[32] A. Kurz. A co-variety-theorem for modal logic. In *Advances in Modal Logic 2*, pages 367–380. CSLI, 2001. papers from the second workshop on "Advances in Modal logic," held in Uppsala, Sweden, 1998.

[33] A. Kurz. Specifying coalgebras with modal logic. *Theor. Comp. Sci.*, 260, 2001.

[34] R. S. Myers. *Rational Coalgebraic Machines in Varieties: Languages, Completeness and Automatic Proofs.* PhD thesis, Imperial College London, 2011.

[35] J. Power and D. Turi. A coalgebraic foundation for linear time semantics. *Electr. Notes Theor. Comput. Sci.*, 29:259–274, 1999.

[36] M. Rößiger. Coalgebras and modal logic. In *CMCS'00*.

[37] M. Rößiger. Languages for coalgebras on datafunctors. In *CMCS'00*, volume 19 of *ENTCS*, 1999.

[38] J. Rutten. Automata and coinduction - an exercise in coalgebra. In *CONCUR'98*.

[39] J. Rutten. Coalgebra, concurrency, and control. Report SEN-R9921, CWI, Amsterdam, 1999.

[40] J. Rutten. Behavioural differential equations: A coinductive calculus of streams, automata, and power series. Report SEN-R0023, CWI, Amsterdam, 2000.

[41] J. Rutten. Universal coalgebra: A theory of systems. *Theor. Comp. Sci.*, 249, 2000.

[42] J. Rutten. Coinductive counting with weighted automata. *J. Autom. Lang. Comb.*, 8, 2003.

[43] J. Rutten. A tutorial on coinductive stream calculus and signal flow graphs. *Theor. Comp. Sci.*, 343, 2005.

[44] J. Rutten and D. Turi. On the foundations of final semantics: Non-standard sets, metric spaces, partial orders. Report CS-R9241, CWI, Amsterdam, 1992.

[45] J. Rutten and D. Turi. Initial algebra and final coalgebra semantics for concurrency. Report CS-R9409, CWI, Amsterdam, 1994.

[46] A. Silva, F. Bonchi, M. M. Bonsangue, and J. J. M. M. Rutten. Quantitative kleene coalgebras. *Inf. Comput.*, 209(5):822–849, 2011.

[47] A. Silva, M. M. Bonsangue, and J. J. M. M. Rutten. Non-deterministic kleene coalgebras. *Logical Methods in Computer Science*, 6(3), 2010.

[48] D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *LICS'97*, 1997.

[49] D. Turi and J. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Math. Structures Comp. Sci.*, 8, 1998.

[50] J. Winter, M. M. Bonsangue, and J. J. M. M. Rutten. Context-free languages, coalgebraically. In *Algebra and Coalgebra in Computer Science - 4th International Conference, CALCO 2011, Winchester, UK, August 30 - September 2, 2011. Proceedings*, pages 359–376, 2011.

[51] J. Winter, M. M. Bonsangue, and J. J. M. M. Rutten. Coalgebraic characterizations of context-free languages. *Logical Methods in Computer Science*, 9(3), 2013.

[52] J. Winter, M. M. Bonsangue, and J. J. M. M. Rutten. Context-free coalgebras. *J. Comput. Syst. Sci.*, 81(5):911–939, 2015.

[53] J. Worrell. On the final sequence of an finitary set functor. *Theor. Comp. Sci.*, 338, 2005.