PhD Topic: Coalgebraic Logic and Distributive Laws

Ruben Turkenburg

April 17, 2022

1 Introduction

Coalgebras have now been studied for a number of decades as a way to investigate systems with some notion of states and transitions. A simple example of a system which can be studied in this way is a traffic light system. In this case, the state of the system is the combination of lights that are on, and the transitions are the usual ones, such as going from green to amber. More complex examples include protocols for communication over networks and computer programs, which manipulate the state of a computer's memory by carrying out the instructions written by a programmer.

By using coalgebras to study such systems, we take an abstract view allowing for very general insights, applicable to many systems at once. When considering these coalgebras, we would often like to show that certain states have different behaviours (in the sense that we can exhibit a difference in their configuration at some point in time). A way to do this is by considering properties of the system which may hold in one state but not another. For the traffic light example, consider the difference between a British and European (at least Dutch) traffic light system. For the first a transition is made from red to red and amber together, whereas in the second the light goes from red straight to green. Thus, a property of the first which is not enjoyed by the second is that in the state signified by the red light being on, a possible following state will have red and amber on simultaneously. This shows that in these two systems, the states where the red light is on do not have the same behaviour. This kind of reasoning is easily applicable to many systems, however we would also like to be able to use the converse: that states which have all the same properties also behave in the same way. Unfortunately, this is not always the case, but in many cases there are ways to extend the system so that we can apply this reasoning while preserving the original states (this is important, as otherwise we might not be able to talk about the behaviour of the states we started with). A current area of interest is to do this "changing of systems" in the realm of coalgebras in a way which is more easily (and more generally) applicable than current methods. This is what I am looking at during this phase of my PhD.

During this work, a tool that arises quite often is distributive laws. These are a generalisation of rules such as

$$x \cdot (y + z) = x \cdot y + x \cdot z \tag{1}$$

$$(w + x) \cdot (y + z) = w \cdot y + w \cdot z + x \cdot y + x \cdot z \tag{2}$$

where we say that multiplication distributes over addition. This kind of interchange between operations appears also in (functional) programming. For example, a well known function on lists is `zip`, which takes a pair of lists and creates a list of pairs each consisting of an element of

the first list and an element of the second list in the input pair. Here, we notice an interchange between lists and pairs. A more complex example, which is of interest in the area of coalgebra, is the combination of what are called the powerset and finite distribution monads. These can be used to model fairly simple transition systems (similar to the traffic light example) and systems where the transitions are taken with a certain probability respectively. If we want to model a situation where both of these behaviours arise simultaneously, a distributive law makes this much simpler. All of these examples can be studied as instances of the same abstract construction, which we explain later.

2 Formal Definitions

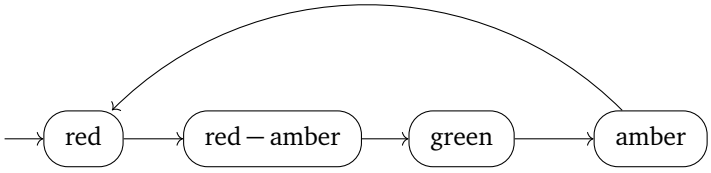
2.1 Coalgebras

The formal definition of a coalgebra is given in the language of category theory (more on this is given in Appendix A, we give some intuition and revisit the traffic light example after stating the definition.

Definition 2.1. *For an endofunctor* $F\colon \mathscr{C} \to \mathscr{C}$ *an* F-coalgebra *is a pair* (X, γ)*, with* X *an object of the category* $\mathscr{C}$ *and* $\gamma\colon X \to FX$ *a morphism in* $\mathscr{C}$*.*

The idea is that the X in this definition is the collection of states in which a system we are studying can be, and the morphism (think of a function) γ tells us how the state of the system can change.

Example 2.2. *If we try to fit the (British) traffic light example into the definition of a coalgebra, we need to start with the collection of possible states. In this case, this is simply a set (and the category* $\mathscr{C}$ *we are working in is the category* **Set** *of sets and functions, but this is not important for the example) which we could write as* $\{\text{red}, \text{red}-\text{amber}, \text{amber}, \text{green}\}$*. Now, we must give a function* $\gamma\colon X \to FX$*, which describes the transitions of a traffic light. For this example, we ignore the endofunctor* F *and simply have a function* $X \to X$ *defined by* $\text{red} \mapsto \text{red}-\text{amber} \mapsto \text{green} \mapsto \text{amber} \mapsto \text{red}$*, where the symbol* $\mapsto$ *can be read as "makes a transition to." It is hopefully already clear how this describes the possible changes of state of a traffic light, but we can also give a diagrammatic representation as follows:*



More generally, where we do not ignore the endofunctor F, we can describe more complex systems. For example, we may want to model the failure of a traffic light, whereby there are additional transitions possible from any state to a new state representing the traffic light being off or other strange behaviour such as all lights being on simultaneously. In that case, we could have a function mapping a state to a set of possible following states, rather than simply a single state.

Well known examples in the field of computer science include: various forms of finite automata; finite state machines; and Turing machines. These can in turn be used to model simple computations, systems where some form of interaction is possible (like communication protocols) and computations of general computers respectively.

2.2 Distributive Laws

This is going to be hard to explain ☺.

Distributive laws in the form used when studying coalgebras are again formulated in terms of category theory. We will give some definitions followed by some intuition.

The most basic form of distributive law, is based on functors.

Definition 2.3. *A* distributive law of an endofunctor $\mathtt{B}$ over an endofunctor $\mathtt{T}$ *is a natural transformation* $\lambda : \mathtt{BT} \Rightarrow \mathtt{TB}$.

We see in this definition some form of interchange of the operations given by $\mathtt{B}$ and $\mathtt{T}$, however it is unfortunately not so easy to fit the examples from earlier into this picture.

Also of interest are cases where the functors involved in a distributive law form what are called monads.

Definition 2.4. *Let $\mathscr{C}$ be a category. A* monad *is a triple* $(\mathtt{T}, \eta, \mu)$ *made up of:* $\mathtt{T} : \mathscr{C} \to \mathscr{C}$ *an endofunctor on $\mathscr{C}$ and* $\eta : 1 \Rightarrow \mathtt{T}$ *and* $\mu : \mathtt{T}^2 \Rightarrow \mathtt{T}$ *natural transformations, such that the following equations hold:* $\mu \circ \mathtt{T}\eta = 1_{\mathtt{T}} = \mu \circ \eta\mathtt{T}$ *and* $\mu \circ \mathtt{T}\mu = \mu \circ \mu\mathtt{T}$.

Of this definition, the most important elements are the natural transformations η and μ. These provide, for any object $\mathtt{X}$ of the category $\mathscr{C}$, morphisms $\eta_{\mathtt{X}} : \mathtt{X} \to \mathtt{TX}$ and $\mu_{\mathtt{X}} : \mathtt{TTX} \to \mathtt{TX}$. We can think of $\mathtt{T}$ as giving some structure based on the object $\mathtt{X}$, for example, if $\mathtt{X}$ is a set we could have a $\mathtt{T}$ so that $\mathtt{TX}$ is the set of subsets of $\mathtt{X}$. We expand on this example now.

Example 2.5. *The endofunctor giving for a set its set of subsets is denoted by $\mathscr{P}$. The most common transformation* $\eta_{\mathtt{X}} : \mathtt{X} \to \mathscr{P}\mathtt{X}$ *is then given by* $x \mapsto \{x\}$ *i.e. we take an element of the set* $\mathtt{X}$*, and form the set containing just that element. The transformation* $\mu_{\mathtt{X}} : \mathscr{P}\mathscr{P}\mathtt{X} \to \mathscr{P}\mathtt{X}$ *which is combined with this* η*, takes a set of subsets of* $\mathtt{X}$ *and returns the union of these sets. For example, applying this to the set* $\{\{w, x\}, \{y, z\}\}$ *yields* $\{w, x, y, z\}$*. We can think of this as a "flattening" operation.*

In general, we can think of the transformations η and μ giving us a way to construct basic elements of the "structured" object $\mathtt{TX}$ from elements of $\mathtt{X}$ and a way to combine elements of the "doubly structured" object $\mathtt{TTX}$ into elements of $\mathtt{TX}$.

Example 2.6. *Perhaps a more approachable example (although one which is actually rather similar to Example 2.5) again comes from functional programming. Here, the object* $\mathtt{X}$ *represents a type of the programming language, and we consider the* $\mathtt{T}$ *forming lists of elements of type* $\mathtt{X}$*. The map* $\eta_{\mathtt{X}} : \mathtt{X} \to \mathtt{TX}$ *constructs from an element* x *of type* $\mathtt{X}$*, the list* $[x]$ *containing only* x*. The map* $\mu_{\mathtt{X}} : \mathtt{TTX} \to \mathtt{TX}$ *is again a "flattening" operation taking a list of lists and combining these lists, e.g.,* $[[1, 2, 3], [3, 4, 5]] \mapsto [1, 2, 3, 3, 4, 5]$.

The definitions of a distributive law of a monad over a monad or monad over an endofunctor are rather involved (so we don't give them here). They start with the same transformation as in Definition 2.3, and extend this with conditions on how this transformation interacts with the morphisms involved in the monad(s). When we have two monads, a corresponding distributive law gives a way to build a new monad:

Example 2.7. *Given a distributive law* $\lambda : \mathtt{BT} \Rightarrow \mathtt{TB}$ *of a monad* $(\mathtt{B}, \eta^{\mathtt{B}}, \mu^{\mathtt{B}})$ *over a monad* $(\mathtt{T}, \eta^{\mathtt{T}}, \mu^{\mathtt{T}})$*, we can construct a monad based on the functor* $\mathtt{TB}$ *obtained by composition. The map* $\eta^{\mathtt{TB}}$ *is constructed as*

$$\mathtt{X} \xrightarrow{\eta^{\mathtt{B}}_{\mathtt{X}}} \mathtt{BX} \xrightarrow{\eta^{\mathtt{T}}_{\mathtt{BX}}} \mathtt{TBX} \tag{3}$$

and the map $\mu^{T}B$ *is constructed as*

$$TBTBX \xrightarrow{T\lambda_{BX}} TTBBX \xrightarrow{TT\mu^{B}_{X}} TTBX \xrightarrow{\mu^{T}_{BX}} TBX \tag{4}$$

These types of constructions are often of interest to computer scientists; they give us a way to build new operations from existing ones, which is very useful.

A Some category theory

Our first definition is of course that of a category.

Definition A.1 (Category)**.** *A category* $\mathscr{C}$ *consists of:*

- *A collection of objects* $\mathrm{Ob}(\mathscr{C})$
- *A collection of morphisms* $\mathrm{Hom}(\mathscr{C})$
- *For each morphism* f *a source* $s(f)$ *and target* $t(f)$
- *For any two morphisms* f, g *with* $t(f) = s(g)$*, a morphism* $g \circ f$
- *For any object* X*, a morphism* id_X

such that the following hold

- $s(g \circ f) = s(f)$ *and* $t(g \circ f) = t(g)$
- $s(\mathsf{id}_X) = X = t(\mathsf{id}_X)$
- *The composition of morphisms is associative:* $(h \circ g) \circ f = h \circ (g \circ f)$
- *The morphisms* id_X *act as units in the following way:* $\mathsf{id}_Y \circ f = f$ *when* $t(f) = Y$ *and* $f \circ \mathsf{id}_X = f$ *when* $s(f) = X$

We will write $X \in \mathscr{C}$ for X an object of $\mathscr{C}$, and $f : X \to Y$ for a morphism of $\mathscr{C}$ with $s(f) = X$ and $t(f) = Y$. We also use $\mathrm{Hom}(X, Y)$ to note the collection of morphisms with source X and target Y. A category is called *small* if $\mathrm{Hom}(X, Y)$ is a set for all objects $X, Y \in \mathscr{C}$. For any category $\mathscr{C}$, there is an *opposite category* $\mathscr{C}^{\mathrm{op}}$ with the same objects and where a morphism $f : X \to Y$ in $\mathscr{C}^{\mathrm{op}}$ is simply a morphism $f : Y \to X$ in $\mathscr{C}$ where the composition $g \circ f$ in $\mathscr{C}^{\mathrm{op}}$ is defined as $f \circ g$ in $\mathscr{C}$.

Next, we define maps between categories called functors, which preserve the structure of the categories involved in the sense that the rules for identity and composition of morphisms are respected. It is also possible to compose functors in the expected fashion. In this way, categories themselves form a category **Cat**, which is one the reasons the objects in a category are defined as forming a collection and not a set.

Definition A.2 (Functor)**.** *A functor* $F : \mathscr{C} \to \mathscr{D}$ *is a map taking objects* $X \in \mathscr{C}$ *to objects* $F(X) \in \mathscr{D}$ *and morphisms* $f : X \to Y$ *in* $\mathscr{C}$ *to morphisms* $F(f) : F(X) \to F(Y)$ *in* $\mathscr{C}$*, such that:*

- $F(\mathsf{id}_X) = \mathsf{id}_{F(X)}$ *for all objects* $X \in \mathscr{C}$
- $F(g \circ f) = F(g) \circ F(f)$ *for morphisms* $f, g \in \mathrm{Hom}(\mathscr{C})$ *such that the composition exists.*

A functor $F : \mathscr{C} \to \mathscr{C}$ is also called an endofunctor. We will often omit the brackets and write Ff and FX instead of $F(f)$ and F(X).

The following example of a functor will often be useful, for example, in later definitions of this section.

Definition A.3. *For an object* $C \in \mathscr{C}$*, the constant functor* $\Delta_C \colon \mathscr{D} \to \mathscr{C}$ *sends any object* $D \in \mathscr{D}$ *to* C *and any morphism in* $\mathscr{D}$ *to* id_C.

We have now seen morphisms between objects of a category and morphisms between categories themselves. Going a level higher, we define morphisms between functors.

Definition A.4 (Natural Transformation)**.** *For functors* $F, G \colon \mathscr{C} \to \mathscr{D}$*, a natural transformation* $\alpha \colon F \Rightarrow G$ *gives for each object* $X \in \mathscr{C}$ *a morphism* $\alpha_X \colon FX \to GX$ *such that for any* $f : X \to Y$ *the following diagram commutes:*

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ \downarrow{\scriptstyle \alpha_X} & & \downarrow{\scriptstyle \alpha_Y} \\ GX & \xrightarrow{Gf} & GY \end{array}$$

It should be clear that we can compose functors with functors. We can also compose natural transformations with natural transformations, but this can be done in two ways. If we have natural transformations $\lambda \colon F \Rightarrow G$ and $\nu \colon G \Rightarrow H$ between functors $F, G, H \colon \mathscr{C} \to \mathscr{D}$, then the composition $\nu\lambda \colon F \Rightarrow H$ is defined component-wise using composition of functors by $(\nu\lambda)_X = \nu_X \lambda_X$. The other form of composition applies if we have a natural transformation $\lambda \colon F \Rightarrow G$ between functors $F, G \colon \mathscr{C} \to \mathscr{D}$ and another natural transformation $\nu \colon H \Rightarrow K$ between functors $H, K \colon \mathscr{D} \to \mathscr{E}$. Then there exists a composition $\nu * \lambda \colon HF \Rightarrow KG$.

It is also possible to compose functors and natural transformations, again in two ways. For $\lambda \colon F \Rightarrow G$ a natural transformation between functors $F, G \colon \mathscr{C} \to \mathscr{D}$ and $H \colon \mathscr{D} \to \mathscr{E}$ another functor, the composition $H\lambda \colon HF \Rightarrow HG$ is defined by $(H\lambda)_X = H(\lambda_X)$. If we instead have a functor $K \colon \mathscr{B} \to \mathscr{C}$ we can form the composition $\lambda K \colon FK \Rightarrow GK$ given by $(\lambda K)_X = \lambda_{KX}$.