Colecture 2: Monads

Aleks Kissinger (and Juriaan Rot)

November 11, 2018

1 Terms

An important kind of algebraic data type is a type of *terms* for a signature Σ. These form the connection between initial algebras of a functor and algebraic structures in the usual sense (groups, rings, etc.).

A *signature* Σ is a collection of *symbols* with *arities*. Symbols are just names (i.e. the names of some collection of functions we care about), and arities tell us how many arguments those function symbols take. For example:

$$\Sigma_G = \{m : 2, i : 1, e : 0\}$$

is a signature. A valid *term* for this signature is any composition of those symbols respecting arities. For example:

$$m(m(i(e), e), e)$$

is a valid term, but $e(m)$ is not.

Signatures are the starting point for defining an algebraic structure. They tell us what operations are allowed. For example, the signature Σ above defines the allowed operations for a *group*, namely: multiplication, inverse, and unit.

The set of terms for a signature is the initial algebra of a functor of the form:

$$F(X) = X^{n_1} + X^{n_2} + \ldots + X^{n_k}$$

where n_i is the arity of the i-th generator. For example, the functor for the group-signature given above is:

$$F(X) = X^2 + X + 1$$

In datatype-notation, this initial algebra is given by:

$$\begin{aligned} \textbf{datatype}\ \mathsf{GTerm} = {} & m\ \textbf{of}\ \mathsf{GTerm} \times \mathsf{GTerm} \\ \mid {} & i\ \textbf{of}\ \mathsf{GTerm} \\ \mid {} & e \end{aligned}$$

Exercise 1.1. Give a functor whose initial algebras are terms for a ring.

The elements of GTerm are terms constructed using the symbols from the group signature Σ_G. However, the elements of this algebra are only *ground terms*, i.e. they have no variables in them, only constants. In particular, we have no way of stating the axioms of a group (which indeed involve variables):

$$m(a, m(b, c)) = m(m(a, b), c)$$

$$m(a, e) = a = m(e, a) \qquad m(a, i(a)) = e = m(i(a), a)$$

We can fix this by adding a parameter, like we had for lists, and pass it to a new constructor called var:

$$\begin{aligned} \textbf{datatype } \mathsf{GTerm}(A) = \ & \mathsf{var} \textbf{ of } A \\ & \mid m \textbf{ of } \mathsf{GTerm} \times \mathsf{GTerm} \\ & \mid i \textbf{ of } \mathsf{GTerm} \\ & \mid e \end{aligned}$$

That is, we take the initial algebra of the functor:

$$F_A(X) = A + X^2 + X + 1 \tag{1}$$

Now, elements of $\mathsf{GTerm}(A)$ look like, e.g.:

$$m(\mathsf{var}(a), m(\mathsf{var}(b), y, \mathsf{var}(c)))$$

where $a, b, c \in A$ are variables taken from an arbitrary set A. Cool! We are one step closer to being able to talk about real algebraic stuff using endofunctors. But before we get there, lets make a couple of observations.

First, if we have an $f : A \to B$, we can build a function

$$\mathsf{GTerm}(f) : \mathsf{GTerm}(A) \to \mathsf{GTerm}(B)$$

much like we did for lists:

$$\begin{tikzcd} A + \mathsf{GTerm}(A)^2 + \mathsf{GTerm}(A) + 1 \arrow[r, dashed] \arrow[d, "{[\mathsf{var}, m, i, e]}"'] & A + \mathsf{GTerm}(B)^2 + \mathsf{GTerm}(B) + 1 \arrow[d, "{[\mathsf{var} \circ f, m, i, e]}"] \\ \mathsf{GTerm}(A) \arrow[r, dashed, "\mathsf{GTerm}(f)"] & \mathsf{GTerm}(B) \end{tikzcd}$$

This function recurses all the way down to variables, then applies f. We can see the associated ML code by listing the recurrence relations:

$$\begin{aligned} & \textbf{fun } \mathsf{GTerm}(f)(m(x, y)) = m(\mathsf{GTerm}(f)(x), \mathsf{GTerm}(f)(y)) \\ & \quad \mid \mathsf{GTerm}(f)(i(x)) = i(\mathsf{GTerm}(f)(x)) \\ & \quad \mid \mathsf{GTerm}(f)(e) = e \\ & \quad \mid \mathsf{GTerm}(f)(\mathsf{var}(a)) = \mathsf{var}(f(a)) \end{aligned}$$

Now, we can take any set A to be the set of variables for $\mathsf{GTerm}(A)$. So, what happens if we take $\mathsf{GTerm}(A)$ itself? Well, we have terms who's "variables" are themselves terms, e.g.

$$m(\mathsf{var}(\boxed{m(e,\mathsf{var}(a))}),\mathsf{var}(\boxed{e})) \in \mathsf{GTerm}(\mathsf{GTerm}(A))$$

Well, this is really just a term in $\mathsf{GTerm}(A)$ again. All we have to do is "flatten" it, i.e. delete one layer of var:

$$m(\mathsf{var}(\boxed{m(e,\mathsf{var}(a))}),\mathsf{var}(\boxed{e})) \mapsto m(m(e,\mathsf{var}(a)),e)$$

As ML, this function looks like:

```
**fun** $\mathsf{flat}(\mathsf{var}(a)) = a$
    $|\ \mathsf{flat}(m(x,y)) = m(\mathsf{flat}(x),\mathsf{flat}(y))$
    $|\ \mathsf{flat}(i(x)) = i(\mathsf{flat}(x))$
    $|\ \mathsf{flat}(e) = e$
```

We can also do the opposite thing. For any $a \in A$, we can get a term just by wrapping it in "var". We'll call this lift:

$$\textbf{fun}\ \mathsf{lift}(a) = \mathsf{var}(a)$$

We can build these using the initial algebra structure as well. But first, note we don't really need the exact definition of F from (1) to construct $\mathsf{GTerm}(A)$, we just need the fact that:

$$F_A(X) := A + F(X)$$

has an initial algebra. Let $F^*(A)$ be the initial algebra of F_A.

Exercise 1.2. Prove that F^* extends to a functor for any F such that F_A has an initial algebra for all $A \in \mathrm{ob}(\mathcal{C})$.

Every initial algebra comes with a morphism. For $F^*(A)$, let's call it:

$$[\mathsf{lift},\mathsf{gens}] : A + F(F^*(A)) \to F^*(A)$$

Boom! We already have lift. It is just part of the initial algebra structure. For the group example, $\mathsf{lift} = \mathsf{var}$ and gens is the rest, all wrapped into a single map:

$$\mathsf{gens} = [m,i,e] : \mathsf{GTerm}(A)^2 + \mathsf{GTerm}(A) + 1 \to \mathsf{GTerm}(A)$$

So, flat should be defined in terms of gens, recursively. By definition, $F^*(F^*(A))$ is the initial algebra of $F_{F^*(A)}$. Hence we can get a function

$$\mathsf{flat}_A : F^*(F^*(A)) \to F^*(A)$$

by giving $F^*(A)$ the structure of an $F_{F^*(A)}$ algebra. That's a little bit mind-bending, but it's actually not too bad if we keep a clear head. We need to complete this square:

$$\begin{tikzcd}
F_{F^*(A)}(F^*(F^*(A))) \arrow[r, dashed] \arrow[d, "a"'] & F_{F^*(A)}(F^*(A)) \arrow[d, "?"] \\
F^*(F^*(A)) \arrow[r, dashed, "\mathsf{flat}"] & F^*(A)
\end{tikzcd}$$

i.e. this square:

$$\begin{tikzcd}
F^*(A) + F(F^*(F^*(A))) \arrow[r, dashed] \arrow[d, "{[\mathsf{lift}, \mathsf{gens}]}"'] & F^*(A) + F(F^*(A)) \arrow[d, "?"] \\
F^*(F^*(A)) \arrow[r, dashed, "\mathsf{flat}"] & F^*(A)
\end{tikzcd}$$

Well, if we already have a term in $F^*(A)$, do nothing with it. Otherwise, we use gens:

$$\begin{tikzcd}
F^*(A) + F(F^*(F^*(A))) \arrow[r, dashed, "F^*(A) + F(\mathsf{flat})"] \arrow[d, "{[\mathsf{lift}, \mathsf{gens}]}"'] & F^*(A) + F(F^*(A)) \arrow[d, "{[\mathsf{id}_{F^*(A)}, \mathsf{gens}]}"] \\
F^*(F^*(A)) \arrow[r, dashed, "\mathsf{flat}"] & F^*(A)
\end{tikzcd}$$

So, that get pretty abstract, so lets see if we did The Right Thing. The commutative square captures the equation:

$$\mathsf{flat} \circ [\mathsf{lift}, \mathsf{gens}] = [\mathsf{id}, \mathsf{gens} \circ F(\mathsf{flat})]$$

which written again in ML-style notation gives us:

$$\begin{aligned}
&\textbf{fun } \mathsf{flat}(\mathsf{lift}(a)) = \mathsf{id}(a) \\
&\quad \mid \mathsf{flat}(\mathsf{gens}(z)) = \mathsf{gens}(F(\mathsf{flat})(z))
\end{aligned}$$

Returning to our group example, $\mathsf{lift} = \mathsf{var}$, so the first line exactly matches our code of flat from before. For the second line, $\mathsf{gens} = [m, i, e]$ so this takes an incoming value z from a coproduct of 3 sets and applies the appropriate constructor:

$$\mathsf{gens}(z) = \begin{cases} m(x,y) & \text{if } z = (x,y) \in \mathsf{GTerm}(A) \times \mathsf{GTerm}(A) \\ i(x) & \text{if } z = x \in \mathsf{GTerm}(A) \\ e & \text{if } z = * \in 1 \end{cases}$$

Similarly, we have: $F(\mathsf{flat}) = \mathsf{flat} \times \mathsf{flat} + \mathsf{flat} + \mathsf{id}_1$, so:

$$F(\mathsf{flat})(z) = \begin{cases} (\mathsf{flat}(x), \mathsf{flat}(y)) & \text{if } z = (x, y) \in \mathsf{GTerm}(A) \times \mathsf{GTerm}(A) \\ \mathsf{flat}(x) & \text{if } z = x \in \mathsf{GTerm}(A) \\ * & \text{if } z = * \in 1 \end{cases}$$

1.1 More natural transformations

One thing we notice about the definitions of flat_A and lift_A is that they don't look directly at the elements of A. They just shuffle them around inside of terms. We saw last time, this corresponds to *polymorphic functions*, or categorically to *natural transformations*.

Recall this definition from last time:

Definition 1.3. For any two functors F, G a natural transformation $\phi : F \to G$ is a family of functions $\phi_A : F(A) \to G(A)$ which makes the follow diagram commute, for all sets A, B and all functions $f : A \to B$:

$$\begin{tikzcd} F(A) \arrow[r, "F(f)"] \arrow[d, "\phi_A"'] & F(B) \arrow[d, "\phi_B"] \\ G(A) \arrow[r, "G(f)"] & G(B) \end{tikzcd}$$

Note in particular that we write the whole family ϕ is a morphism going from the functor F to the functor G. This seems to suggest there is a *category* whose objects are functors and whose morphisms are natural transformations. This is indeed the case:

Definition 1.4. The category $[\mathcal{C}, \mathcal{D}]$ has as objects functors $F : \mathcal{C} \to \mathcal{D}$ and as morphisms natural transformations.

Exercise 1.5. Show that $[\mathcal{C}, \mathcal{D}]$ is a category. What are its identities? What is composition?

We can show that the families of functions lift_A and flat_A yield the following natural transformations:

$$\mathsf{lift} : \mathsf{Id} \to F^*$$
$$\mathsf{flat} : F^* \circ F^* \to F^*$$

Note everything in sight is an endofunctor on Set, so both of the above natural transformations are morphisms in the category [Set, Set].

We have already (secretly) shown that lift is a natural transformation in exercise 1.2. Much like we did with the examples from last time, we should define $F^*(f)$ as:

$$\begin{tikzcd}
A + F(F^*(A)) \arrow[r, dashed, "\mathrm{id}_A + F(F^*(f))"] \arrow[d, "{[\mathsf{lift}, \mathsf{gens}]}"'] & A + F(F^*(B)) \arrow[d, "{[\mathsf{lift} \circ f, \mathsf{gens}]}"] \\
F^*(A) \arrow[r, dashed, "F^*(f)"] & F^*(B)
\end{tikzcd}$$

Just reading the A-part of this commutative square gives:

$$\begin{tikzcd}
A \arrow[r, "\mathrm{id}_A"] \arrow[d, "\mathsf{lift}"'] & A \arrow[d, "\mathsf{lift} \circ f"] \\
F^*(A) \arrow[r, "F^*(f)"] & F^*(B)
\end{tikzcd}$$

Shifting the f around to the top, we can write the same equation as this commutative square:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "\mathsf{lift}"'] & B \arrow[d, "\mathsf{lift}"] \\
F^*(A) \arrow[r, "F^*(f)"] & F^*(B)
\end{tikzcd}$$

which is exactly naturality for lift! flat is a bit harder, due to the recursive definition. So, we'll make this an optional exercise:

Exercise* 1.6. (Bonus) Prove that the following diagram commutes:

$$\begin{tikzcd}
F^*(F^*(A)) \arrow[r, "F^*(F^*(f))"] \arrow[d, "\mathsf{flat}"'] & F^*(F^*(B)) \arrow[d, "\mathsf{flat}"] \\
F^*(A) \arrow[r, "F^*(f)"] & F^*(B)
\end{tikzcd}$$

In addition to naturality, we notice that lift and flat interact well with each other. For instance, if a lift a term then flatten it, I get back where I started:

$$m(e, \mathsf{var}(a)) \mapsto \mathsf{var}(\boxed{m(e, \mathsf{var}(a))}) \mapsto m(e, \mathsf{var}(a))$$

I can write this as a commutative diagram:

$$\begin{tikzcd}
F^*(A) \arrow[r, "\mathrm{id}"] \arrow[d, "\mathsf{lift}_{F^*(A)}"'] & F^*(A) \\
F^*(F^*(A)) \arrow[ur, "\mathsf{flat}_A"'] &
\end{tikzcd}$$

Similarly, if I lift all the var's in a term, via $F^*(\mathsf{lift})$, then flat, I get back where I started:

$$m(e, \mathsf{var}(a)) \mapsto m(e, \mathsf{var}(\boxed{\mathsf{var}(a)})) \mapsto m(e, \mathsf{var}(a))$$

I can also write this as a commutative diagram:

$$\begin{tikzcd}
F^*(A) \arrow[r, "\mathrm{id}"] \arrow[d, "F^*(\mathsf{lift}_A)"'] & F^*(A) \\
F^*(F^*(A)) \arrow[ur, "\mathsf{flat}_A"'] &
\end{tikzcd}$$

Finally, if I have "triple terms", there are two ways I can flatten them. I can first recurse into all the var's and flatten those, using $F^*(\mathsf{flat}_A)$, then flatten the whole term using flat_A:

$$m(e, \mathsf{var}(\boxed{m(\mathsf{var}(\boxed{e}), \mathsf{var}(\boxed{\mathsf{var}(a)}))}))$$

$$\mapsto m(e, \mathsf{var}(\boxed{m(e, \mathsf{var}(a))}))$$

$$\mapsto m(e, m(e, \mathsf{var}(a)))$$

...or I can first flatten at the top level using $\mathsf{flat}_{F^*(A)}$, then flatten again using flat_A:

$$m(e, \mathsf{var}(\boxed{m(\mathsf{var}(\boxed{e}), \mathsf{var}(\boxed{\mathsf{var}(a)}))}))$$

$$\mapsto m(e, m(\mathsf{var}(\boxed{e}), \mathsf{var}(\boxed{\mathsf{var}(a)})))$$

$$\mapsto m(e, m(e, \mathsf{var}(a)))$$

These should of course be the same:

$$\begin{tikzcd}
F^*(F^*(F^*(A))) \arrow[r, "\mathsf{flat}_{F^*(A)}"] \arrow[d, "F^*(\mathsf{flat}_A)"'] & F^*(F^*(A)) \arrow[d, "\mathsf{flat}_A"] \\
F^*(F^*(A)) \arrow[r, "\mathsf{flat}_A"] & F^*(A)
\end{tikzcd}$$

2 Monads

A monad is an endofunctor that comes with a natural transformation like lift (typically called η) and a natural transformation like flat (typically called μ), satisfying the commutative diagrams we have already seen.

Definition 2.1. A *monad* is an endofunctor $M : \mathcal{C} \to \mathcal{C}$ along with two natural transformations $\mu : M \circ M \to M$ and $\eta : \mathrm{Id}_{\mathcal{C}} \to M$ such that for all $A \in \mathrm{ob}(\mathcal{C})$, the following diagrams commute:

$$\begin{tikzcd}
M(M(M(A))) \arrow[r, "\mu_{M(A)}"] \arrow[d, "\mu_A"'] & M(M(A)) \arrow[d, "\mu_A"] \\
M(M(A)) \arrow[r, "\mu_A"] & M(A)
\end{tikzcd}$$

$$\begin{tikzcd}
M(A) \arrow[r, "\mathrm{id}"] \arrow[d, "\eta_{M(A)}"'] & M(A) \\
M(M(A)) \arrow[ur, "\mu_A"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
M(A) \arrow[r, "\mathrm{id}"] \arrow[d, "M(\eta_A)"'] & M(A) \\
M(M(A)) \arrow[ur, "\mu_A"'] &
\end{tikzcd}$$

Exercise 2.2. Show that the powerset endofunctor $\mathcal{P} : \mathsf{Set} \to \mathsf{Set}$ is a monad. What are μ and η?

Exercise 2.3. Show that $F(X) = X \times \mathbb{N}$ admits at least two distinct monad structures.

The monad we met in the previous section is called the *free monad* of an endofunctor. We'll (hopefully) hear a bit more about this later.

A particularly handy monad for programmers is the "option" or "exception" monad. This is a monad for the functor: $\mathsf{Option}(A) = A + 1$. This is particularly handy for defining partial functions, i.e. functions that could fail on some inputs. Thus, it has one constructor to give a value (in the case where the function was successful), and one to indicate failure:

$$[\mathsf{yay}, \mathsf{boo}] : A + 1 \to \mathsf{Option}(A)$$

So, in **datatype** notation, it looks like this:

$$\begin{aligned}\textbf{datatype}\ \mathsf{Option}(A) &= \mathsf{yay}\ \textbf{of}\ A \\ &\mid\ \mathsf{boo}\end{aligned}$$

This gives us a functor via:

$$\begin{aligned}&\textbf{fun}\ \mathsf{Option}(f)(\mathsf{yay}(x)) = f(x)\\ &\quad\mid \mathsf{Option}(f)(\mathsf{boo}) = \mathsf{boo}\end{aligned}$$

and a monad via:

$$\begin{aligned}&\textbf{fun}\ \mathsf{eta}(a) = \mathsf{yay}(a)\\ &\textbf{fun}\ \mathsf{mu}(\mathsf{boo}) = \mathsf{boo}\\ &\quad\mid \mathsf{mu}(\mathsf{yay}(x)) = x\end{aligned}$$

So, eta lifts an element of A to a "successful" element of $\mathsf{Option}(A)$, whereas mu flattens two levels of Option, propagating success/failure to outside:

$$\mathsf{mu} :: \ \mathsf{boo} \mapsto \mathsf{boo},\ \ \mathsf{yay}(\mathsf{boo}) \mapsto \mathsf{boo},\ \ \mathsf{yay}(\mathsf{yay}(a)) \mapsto \mathsf{yay}(a)$$

Now, a function that might fail is of type $f : A \to \mathsf{Option}(B)$. Consider, for example, adding plus 1 and minus 1 functions to natural numbers, which allow for failure:

$$\begin{aligned}&\textbf{fun}\ \mathsf{plus1}(x) = \mathsf{yay}(\mathsf{suc}(x))\\ &\textbf{fun}\ \mathsf{minus1}(\mathsf{suc}(x)) = \mathsf{yay}(x)\\ &\quad\mid \mathsf{minus1}(\mathsf{zero}) = \mathsf{boo}\end{aligned}$$

Then, these give us functions of the form $\mathbb{N} \to \mathsf{Option}(\mathbb{N})$, which we can "compose" using the monad structure:

$$\begin{aligned}&\textbf{infix}\ \mathsf{THEN}\\ &\ \textbf{fun}\ (f\ \mathsf{THEN}\ g) = \mathsf{mu} \circ \mathsf{Option}(g) \circ f\end{aligned}$$

Now, we can chain together things that might fail, e.g.

$$\textbf{val}\ f = \mathsf{minus1}\ \mathsf{THEN}\ \mathsf{plus1}\ \mathsf{THEN}\ \mathsf{plus1}$$

Then, we have:

$$\begin{aligned}f(\mathsf{suc}(\mathsf{zero})) &\mapsto \mathsf{yay}(\mathsf{suc}(\mathsf{suc}(\mathsf{zero})))\\ f(\mathsf{zero}) &\mapsto \mathsf{boo}\end{aligned}$$

This style of composition using μ is called *Kleisli composition*. It is in fact normal composition, just in a different category!

Definition 2.4. The *Kleisli category* $\mathsf{Kl}(M)$ of a monad M is defined as follows:

- *objects* are the original objects $\mathrm{ob}(\mathcal{C})$ of $\mathcal{C}$,
- *morphisms* $(\widehat{f} : C \to D) \in \mathsf{Kl}(M)$ are morphisms $(f : C \to M(D)) \in \mathcal{C}$,
- *composition* is given by *Kleisli composition*. For $f : A \to M(B)$ and $g : B \to M(C)$, we have:
$$\widehat{g} \circ \widehat{f} := \mu_C \circ M(g) \circ f$$
- *identities* are given by η. That is, $(\mathrm{id}_A : A \to A) \in \mathsf{Kl}(M)$ is given by $(\eta_A : A \to M(A)) \in \mathcal{C}$.

Exercise 2.5. What is the Kleisli category of Option? What is the Kleisli category of $\mathcal{P}$?