Monads and Interaction

Tarmo Uustalu, Reykjavik University

OPLSS 2021, Eugene, OR, 14–26 June 2021

This course

- We will talk about functional computation with effects ("impure" functional computation),
  i.e., so functions are there not only to return values, but do other things along the way (talk to the environment, "world").
- E.g., nondeterministic choice, input/output, manipulation of (external) store.

- The classic approach by Moggi is based on monads, refined by Plotkin and Power (finitary monads = algebraic theories).

- Here we add an explicit discussion of running such computations.
- They are "helpless" alone, without an environment and a communication protocol with it.

- Today: a start on monads.

Prerequisites

- Not sure what level you will find this course is.
- Intermediate?! :-)
- This is necessary:
  - basics of functional programming
  - categories, functors, natural transformations
  - Cartesian (closed) categories, coproducts
  - initial algebras, final coalgebras
- This will help (it's quite a bit to ask, but then none of this is central):
  - adjunctions
  - ends, coends
  - (symmetric) monoidal (closed) categories
- All examples work in **Set**, but almost all are more general.
- Text in gray is additional information for those that have an advanced background; it is not needed to follow the main material.

Warm-up

Some pictures without words

$$\frac{Z \to Y \quad Y \to X}{Z \to X}$$

$$\frac{W \to V \quad V \to Z \quad Z \to Y \quad Y \to X}{W \to X}$$

$$\dfrac{\dfrac{\dfrac{Y \to 1+1 \quad \overline{Y \to Y}}{Y \to (1+1) \times Y}}{Y \to Y + Y} \qquad \dfrac{Y \to X \quad Y \to X}{Y + Y \to X}}{Y \to X}$$

Some pictures without words

$$\frac{Z \to E + Y \quad Y \to E + X}{Z \to E + X} \; ?$$

$$\frac{Z \to \mathsf{List}\, Y \quad Y \to \mathsf{List}\, X}{Z \to \mathsf{List}\, X} \; ?$$

$$\frac{Z \to \mathsf{List}\, Y \quad Y \to \mathsf{List}\, X}{Z \to X} \; ??$$

$$\frac{Z \to E + Y \quad Y \to \mathsf{List}\, X}{Z \to \mathsf{List}\,(E + X)} \; ??$$

Some pictures without words

$$\dfrac{Z \to E + Y \qquad \dfrac{Y \to E + X}{E + Y \to E + (E + X)} \quad \dfrac{}{E + (E + X) \to E + X}\,\mu}{Z \to E + X}$$

$$\dfrac{Z \to \mathsf{List}\, Y \qquad \dfrac{Y \to \mathsf{List}\, X}{\mathsf{List}\, Y \to \mathsf{List}\,(\mathsf{List}\, X)} \quad \dfrac{}{\mathsf{List}\,(\mathsf{List}\, X) \to \mathsf{List}\, X}\,\mu}{Z \to \mathsf{List}\, X}$$

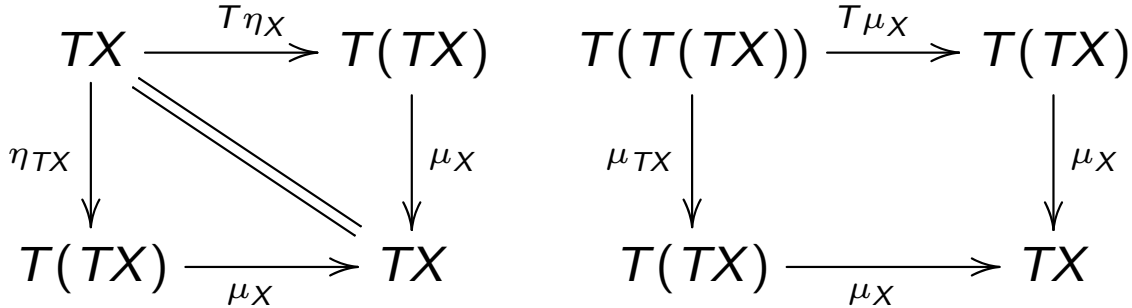
$$\dfrac{\dfrac{Z \to \mathsf{List}\, Y \qquad \dfrac{Y \to \mathsf{List}\, X}{\mathsf{List}\, Y \to \mathsf{List}\,(\mathsf{List}\, X)} \quad \dfrac{}{\mathsf{List}\,(\mathsf{List}\, X) \to \mathsf{List}\, X}\,\mu}{Z \to \mathsf{List}\, X} \qquad \dfrac{X \text{ a monoid}}{\mathsf{List}\, X \to X}}{Z \to X}$$

Monads and their Kleisli categories

Monads

- A *monad* on a category $\mathcal{C}$ is given by a
  - a functor $T : \mathcal{C} \to \mathcal{C}$,
  - a natural transformation $\eta : \mathrm{Id}_{\mathcal{C}} \dot{\to} T$ (the *unit*),
  - a natural transformation $\mu : T \cdot T \dot{\to} T$ (the *multiplication*)

  such that



- Here, $\mathrm{Id}_{\mathcal{C}}$ is the identity functor and $\cdot$ the composition of functors: $\mathrm{Id}_{\mathcal{C}} X = X$ and $(F \cdot G)X = F(GX)$.

- This definition says that monads are monoids in the (strict) monoidal category $([\mathcal{C}, \mathcal{C}], \mathrm{Id}_{\mathcal{C}}, \cdot)$.

FP intuition

- $\mathcal{C}$ – types and functions

- A "computation" is a process that can do various things (talk to the "environment" to get help) and may eventually finish and return a value.
- (T, η, μ) –
  a particular "notion" of computation
  (closed under just returning and sequential composition)

- X – values (of some type X)
- TX – computations of values of type X
- $T(TX)$ – computations of computations of values type X

- $\eta_X : X \to TX$ –
  turns a value into the computation just returning this value
- $\mu_X : T(TX) \to TX$ –
  turns a computation of computations into their sequence

Exceptions monads

- Suppose $\mathcal{C}$ has finite coproducts.
- The exceptions monad for an object E (of *exceptions*) is:
  - $TX = E + X$
  - $\eta_X = X \xrightarrow{\text{inr}} E + X$

    $= X \xrightarrow{\lambda^+} 0 + X \xrightarrow{?+\text{id}} E + X$
  - $\mu_X = E + (E + X) \xrightarrow{[\text{inl},\text{id}]} E + X$

    $= E + (E + X) \xrightarrow{\alpha^+} (E + E) + X \xrightarrow{\nabla+\text{id}} E + X$
- (These η, μ are the only monad structure on this functor T.)
- Computations here are of two forms:
  - $\text{inl}\, e$ – a process that just raises some exception e
  - $\text{inr}\, x$ – a process that just returns some value x

An alternative to monads: Kleisli triples

- A more FP friendly definition is this.

- A *Kleisli triple* (*monad in extension form, no-iteration form*) is given by
  - an object mapping $T : |\mathcal{C}| \to |\mathcal{C}|$,
  - a family of maps $\eta_X : X \to TX$ indexed by $X \in |\mathcal{C}|$,
  - a family of maps $(-)^\star_{X,Y} : \mathcal{C}(Y, TX) \to \mathcal{C}(TY, TX)$ indexed by $X, Y \in |C|$ (the *Kleisli extension* operation)

  such that
  - $k^\star \circ \eta_Y = k$ for $k : Y \to TX$,
  - $\eta^\star_X = \mathrm{id}_{TX}$,
  - $(\ell^\star \circ k)^\star = \ell^\star \circ k^\star : TZ \to TX$ for $k : Z \to TY$, $\ell : Y \to TX$

- Functoriality of T, naturality of η, $(-)^\star$ are not required, but follow.
- So there are only 3 equations instead of 7 for a monad.

FP intuition

- $k^\star : TY \to TX$ for given $k : Y \to TX$ –

  turns a computation of values of Y
  into a computation of values of X

  by replacing returning a value y of Y
  with continuing as the computation $k\,y$ of values of X

Exceptions Kleisli triple

- Recall that the exceptions monad for E had
  - $\mu_X = \ E + (E + X) \xrightarrow{[\mathsf{inl},\mathsf{id}]} E + X$
- The exceptions Kleisli triple has
  - $k^\star = \ E + Y \xrightarrow{[\mathsf{inl},k]} E + X \ \ \text{for } k : Y \to E + X$

Monads = Kleisli triples

- Monads and Kleisli triples with the same $T : |\mathcal{C}| \to |\mathcal{C}|$ and η are in a bijection:

- Given a monad, one obtains a Kleisli triple by
  - $k^\star = \; TY \xrightarrow{Tk} T(TX) \xrightarrow{\mu_X} TX$ for $k : Y \to TX$
- A Kleisli triple is turned into a monad by
  - $Tf = \left(Y \xrightarrow{f} X \xrightarrow{\eta_X} TX \right)^\star : TY \to TX$ for $f : Y \to X$,
  - $\mu_X = \left(TX \xrightarrow{\mathrm{id}_{TX}} TX \right)^\star : T(TX) \to TX$

- The Yoneda lemma is the main ingredient here,

$$\frac{T(TX) \to TX}{\mathcal{C}(Y, TX) \to \mathcal{C}(TY, TX) \text{ nat. in } Y}$$

  but there is more going on.

Monads of Haskell = **strong** Kleisli triples

- Haskell used to have a type class `Monad` like this:

```
class Monad t where
  return :: x -> t x
  (>>=) :: t y -> (y -> t x) -> t x
```

- This class formalizes *strong Kleisli triples*.
- Differently from a (non-strong) Kleisli triple, instead of a family of maps

$$(-)^{\star}_{X,Y} : \mathcal{C}(Y, TX) \to \mathcal{C}(TY, TX) \text{ in } \mathbf{Set}$$

  a strong Kleisli triple on a CCC (more generally a MCC) $\mathcal{C}$ has a family of maps

$$\mathrm{iext}_{X,Y} : Y \Rightarrow TX \to TY \Rightarrow TX \text{ in } \mathcal{C}$$

- (I write $Y \Rightarrow X$ for the exponential object (internal hom).)

- In categories with unique strengths like **Set**, strong functors/monads are "the same" as functors/monads.
- In general categories, strong functors/monads are more special.

Exceptions monads in Haskell

- Exceptions type transformers are instances of `Monad`.

```
class Monad t where
  return :: x -> t x
  (>>=) :: t y -> (y -> t x) -> t x

data Either e x = Left e | Right x

instance Monad (Either e) where
  return x = Right x
  Left  e >>= _ = Left e
  Right y >>= k = k y
```

Monads in Haskell ctd.

- Since GHC 7.10, `Monad` is a subclass of `Applicative` (= strong lax monoidal endofunctors) and that in turn is a subclass of `Functor` (= strong functors).

- This is conceptually problematic since the most "natural" applicative structure on a functor may differ from the one induced by the "natural" monad structure.

- It is misleading since only strong monads are canonically strong applicatives.

- It is problematic for software engineering since a type transformer must be made an instance of `Functor` (and even `Applicative`) before can be made an instance of `Monad`.

  This defeats the whole point of using the Kleisli form.

Kleisli category of a monad

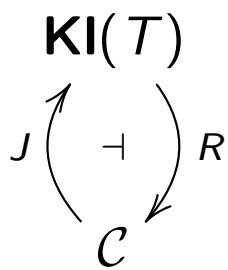
- A monad T on a category $\mathcal{C}$ induces a category $\mathbf{Kl}(T)$ called the *Kleisli category* of T defined by
  - an object is an object of $\mathcal{C}$,
  - a map of from Y to X is a map of $\mathcal{C}$ from Y to TX,
  - $\mathrm{id}_X^T = X \xrightarrow{\eta_X} TX$,
  - $\ell \circ^T k = Z \xrightarrow{k} TY \xrightarrow{T\ell} T(TX) \xrightarrow{\mu_X} TX$, with $\ell^\star : T(TX) \to TX$

    for $k : Z \to^T Y$, $\ell : Y \to^T X$

- From $\mathcal{C}$ there is an identity-on-objects functor J to $\mathbf{Kl}(T)$, defined on maps by
  - $Jf = Y \xrightarrow{f} X \xrightarrow{\eta_X} TX$ for $f : Y \to X$
- If η is mono, then J is faithful.

Kleisli adjunction

- In the opposite direction of $J : \mathcal{C} \to \mathbf{Kl}(T)$, there is a functor $R : \mathbf{Kl}(T) \to \mathcal{C}$ defined by
  - $RX = TX$,
  - $Rk = \; TY \xrightarrow{k^\star} TX$ for $k : Y \to^T X$.
- R is right adjoint to J.



$$\frac{\overbrace{Y}^{JY} \to^T X}{Y \to \underbrace{TX}_{RX}}$$

- Importantly, $R \cdot J = T$. Indeed,
  - $R(JX) = TX$,
  - if $f : Y \to X$, then $R(Jf) = (\eta_X \circ f)^\star = Tf$.
- Moreover, the unit of the adjunction is η.

FP intuition

- $\mathcal{C}$ – types and (pure) functions
- $\mathbf{KI}(T)$ – types and impure functions;
  instead of returning a value of X, they return a computation of values of X
- Jf – the function f turned into a (trivially) impure function

Reader monads

- Suppose $\mathcal{C}$ is Cartesian closed.
- The *reader monad* for an object S (of *readable states*) is:
  - $TX = S \Rightarrow X$
  - $\eta_X : X \to S \Rightarrow X$
    $\eta\, x = \lambda s.\, x$
  - $\mu_X : S \Rightarrow S \Rightarrow X \to S \Rightarrow X$
    $\mu\, f = \lambda s.\, f\, s\, s$
- (These η, μ are the only monad structure on this T.)
- This example generalizes to any monoidal closed category with a given comonoid object.
- In a CCC, any object S comes with a unique comonoid structure given by $!_S : S \to 1$, $\Delta_S : S \to S \times S$.

Writer monads

- Suppose $\mathcal{C}$ has finite products.
- The *writer monad* for a monoid object $(P, \mathsf{o}, \oplus)$ (of *updates, trivial update, composition of updates*) is:
  - $TX = P \times X$
  - $\eta_X : X \to P \times X$
    $\eta\, x = (\mathsf{o}, x)$
  - $\mu_X : P \times (P \times X) \to P \times X$
    $\mu\,(p, (p', x)) = (p \oplus p', x)$
- (Monad structures η, μ on this functor T are in a bijection with monoid structures o, $\oplus$ on the object P.)
- E.g,. we can take $P = \mathsf{Nat}$, $\mathsf{o} = 0$, $\oplus = +$.
- This example generalizes to any monoidal category with a given monoid object.
- It then subsumes the exceptions monads example.
  There, $\mathcal{C}$ is coCartesian monoidal and any object E carries a unique monoid structure $?_E : 0 \to E$ and $\nabla_E : E + E \to E$.

State monads

- Suppose $\mathcal{C}$ is Cartesian closed.
- The *state monad* for an object S (of *readable/overwritable states*) is this:
  - $T\,X = S \Rightarrow S \times X$
  - $\eta_X : X \to S \Rightarrow S \times X$
    $\eta\,x = \lambda s.\,(s, x)$
  - $\mu_X : S \Rightarrow S \times (S \Rightarrow S \times X) \to S \Rightarrow S \times X$
    $\mu\,f = \lambda s.\,\mathsf{let}\ (s', g) = f\ s\ \mathsf{in}\ g\ s'$

- This example generalizes to any monoidal closed category.

List monads

- Suppose $\mathcal{C}$ is distributive, has list objects.
- This the ordinary list monad
  (for a "notion" of nondeterministic computation where any "little" choices it makes to reach outcomes are not recorded, but the "left-to-right" order and multiplicity of the outcomes is):
  - $TX = \mathsf{List}\, X$,
  - $\eta\, x = [x]$,
  - $\mu\, xss = \mathsf{concat}\, xss$.
- It is not the only list monad. Here is another.
  - $TX = \mathsf{List}\, X$,
  - $\eta\, x = [x]$,
  - $\mu\, xss = \begin{cases} [] & \text{if exists null } xss \\ \mathsf{concat}\, xss & \text{otherwise} \end{cases}$
- In fact, there are infinitely many list monads, some very crazy.

Free functor-algebras monads

- Suppose $\mathcal{C}$ has coproducts and the relevant initial algebras.
- This monad delivers (carriers of) free algebras of a functor F:
  - $TX = \mu Z.\, X + F\, Z$ $\qquad (TX \cong^{\mu} X + F(TX))$
    (F-branching trees with X-labelled leaves)
  - $\eta\, x = \mathsf{in}\,(\mathsf{inl}\, x)$
    (turns a value into a tree that is just a leaf)
  - $\mu\,(\mathsf{in}\,(\mathsf{inl}\, t)) = t$
    $\mu\,(\mathsf{in}\,(\mathsf{inr}\, tts)) = \mathsf{in}\,(\mathsf{inr}\,(F\,\mu\, tts))$
    (flattens a tree with leaves labelled with trees into a tree)

- The exceptions monad arises as a special case $FX = E$.

- For $FX = X \times X$, we get
  $TX = \mu Z.\, X + Z \times Z$
  (binary leaf-labelled trees)
  for a notion of nondeterministic computation where the computation makes binary choices and they are all recorded;
  "intensional" nondeterminism
- The nonempty list monad is a quotient of this monad.

Free functor-algebras monads

- For $FX = (S \Rightarrow X) + (S \times X)$, we get
  $TX = \mu Z.\, X + (S \Rightarrow Z) + (S \times Z)$
  – an "intensional" state monad.
- The ordinary ("extensional") state monad is a quotient of this monad.

Continuation monad

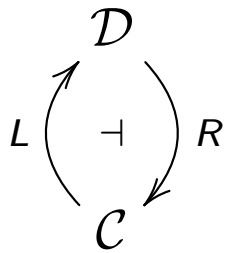
- Suppose $\mathcal{C}$ is Cartesian closed.
- The *continuation monad* for an object R (of *answers*) is
  - $TX = (X \Rightarrow R) \Rightarrow R$
  - $\eta_X : X \to (X \Rightarrow R) \Rightarrow R$
    $\eta\, x = \lambda k.\, k\, x$
  - $\mu_X : (Y \Rightarrow R) \Rightarrow R \to (X \Rightarrow R) \Rightarrow R$
    $\mu\, f = \lambda k.\, f\, (\lambda g.\, g\, k)$
- This example generalizes to any symmetric closed or biclosed category.

- Suppose $\mathcal{C}$ has small products (and is locally small).
- A variation of the continuation monad (the *external continuation monad*) is
  - $TX = \mathcal{C}(X, R) \pitchfork R$

  ($I \pitchfork R$ is the product of I many copies of R, i.e., $\prod_{i \in I} R$, for I a set and $R \in |\mathcal{C}|$.)
- For $\mathcal{C} = \mathbf{Set}$, the two monads are isomorphic.

Monads from adjunctions

- Any adjunction gives rise to a monad.
- Given an adjunction



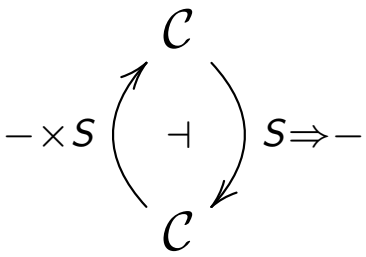
$$\frac{LY \to X}{Y \to RX}$$

the endofunctor $T = R \cdot L$ on $\mathcal{C}$ carries a monad structure with η the unit of the adjunction.

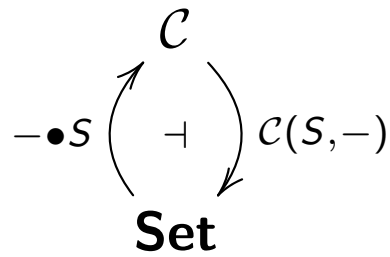
- Adjunctions so related to a monad are called its resolutions.
- Resolutions of a monad form a category.
- The Kleisli adjunction is a resolution. It is the initial object of this category.

State and continuation monads

- The state monads for S arise from the adjunction



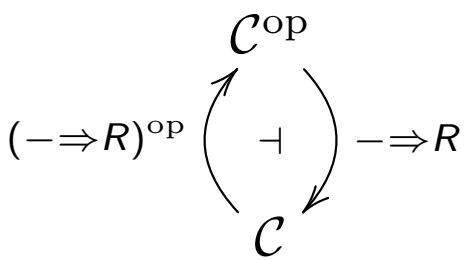
$$\frac{X \times S \to Y}{X \to S \Rightarrow Y}$$



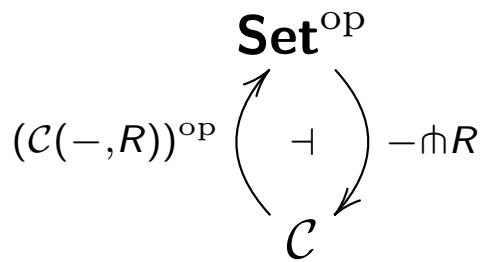
$$\frac{I \bullet S \to Y}{I \to \mathcal{C}(S, Y)}$$

($I \bullet S$ is the coproduct of I many copies of S, i.e., $\coprod_{i \in I} S$, for I a set and $S \in |\mathcal{C}|$.)

- The continuation monads for R arise from the adjunctions



$$\frac{\dfrac{X \Rightarrow R \leftarrow Y}{Y \to X \Rightarrow R}}{X \to Y \Rightarrow R}$$



$$\frac{\dfrac{\mathcal{C}(X, R) \leftarrow I}{I \to \mathcal{C}(X, R)}}{X \to I \pitchfork R}$$