Lecture 16

Monads

Used in Haskell to abstract generic aspects of computation (return a value, sequencing) and to encapsulate effectful code.

Concept imported into functional programming from category theory, first for its denotational semantics by Moggi and then for its practice by Wadler.

Monads

Used in Haskell to abstract generic aspects of computation (return a value, sequencing) and to encapsulate effectful code.

Concept imported into functional programming from category theory, first for its denotational semantics by Moggi and then for its practice by Wadler.

Here, a quick overview of:

- Moggi's computational λ-calculus and its categorical semantics using (strong) monads
- monads and adjunctions

Computational Lambda Calculus (CLC)

CLC extends STLC with new types, terms and equations...

Types: $A, B, \ldots ::=$ STLC types, plus

$\mathtt{T}(A)$ type of "computations" of values of type A

Terms: $s, t, \ldots ::=$ STLC terms, plus

$\mathtt{return}\, t$ trivial computation

$\mathtt{do}\{x \leftarrow s; t\}$ sequenced computation (binds free x in t)

As for STLC, we identify CLC syntax trees up to α-equivalence, where $=_\alpha$ is extended by the rules

$$\frac{t =_\alpha t'}{\mathtt{return}\, t =_\alpha \mathtt{return}\, t'} \quad \text{and} \quad \frac{s =_\alpha s' \qquad (y\ x) \cdot t =_\alpha (y\ x') \cdot t' \qquad y \text{ does not occur in } \{s, s', x, x', t, t'\}}{\mathtt{do}\{x \leftarrow s; t\} =_\alpha \mathtt{do}\{x' \leftarrow s'; t'\}}$$

Computational Lambda Calculus (CLC)

CLC extends STLC with new types, terms and equations…

Types: $A, B, \ldots ::=$ STLC types, plus

$\mathtt{T}(A)$	type of "computations" of values of type A

Terms: $s, t, \ldots ::=$ STLC terms, plus

$\mathtt{return}\, t$	trivial computation
$\mathtt{do}\{x \leftarrow s; t\}$	sequenced computation (binds free x in t)

Typing rules:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathtt{return}\, t : \mathtt{T}(A)}\ (\text{VAL}) \qquad \frac{\Gamma \vdash s : \mathtt{T}(A) \qquad \Gamma, x : A \vdash t : \mathtt{T}(B)}{\Gamma \vdash \mathtt{do}\{x \leftarrow s; t\} : \mathtt{T}(B)}\ (\text{SEQ})$$

Equations…

CLC equations

Extend STLC $\beta\eta$-equality ($\Gamma \vdash s =_{\beta\eta} t : A$) to a relation $\Gamma \vdash s = t : A$ by adding the following rules:

$$\frac{\Gamma \vdash s : A \qquad \Gamma, x : A \vdash t : \mathtt{T}(B)}{\Gamma \vdash \mathtt{do}\{x \leftarrow \mathtt{return}\, s; t\} = t[s/x] : \mathtt{T}(B)}$$

$$\frac{\Gamma \vdash t : \mathtt{T}(A)}{\Gamma \vdash t = \mathtt{do}\{x \leftarrow t; \mathtt{return}\, x\} : \mathtt{T}(A)}$$

$$\frac{\Gamma \vdash s : \mathtt{T}(A) \qquad \Gamma, x : A \vdash t : \mathtt{T}(B) \qquad \Gamma, y : B \vdash u : \mathtt{T}(C)}{\Gamma \vdash \mathtt{do}\{y \leftarrow \mathtt{do}\{x \leftarrow s; t\}; u\} = \mathtt{do}\{x \leftarrow s; \mathtt{do}\{y \leftarrow t; u\}\}}$$

CLC equations

Extend STLC $\beta\eta$-equality ($\Gamma \vdash s =_{\beta\eta} t : A$) to a relation $\Gamma \vdash s = t : A$ by adding the following rules:

$$\frac{\Gamma \vdash s : A \qquad \Gamma, x : A \vdash t : \mathtt{T}(B)}{\Gamma \vdash \mathtt{do}\{x \leftarrow \mathtt{return}\, s; t\} = t[s/x] : \mathtt{T}(B)}$$

$$\frac{\Gamma \vdash t : \mathtt{T}(A)}{\Gamma \vdash t = \mathtt{do}\{x \leftarrow t; \mathtt{return}\, x\} : \mathtt{T}(A)}$$

$$\frac{\Gamma \vdash s : \mathtt{T}(A) \qquad \Gamma, x : A \vdash t : \mathtt{T}(B) \qquad \Gamma, y : B \vdash u : \mathtt{T}(C)}{\Gamma \vdash \mathtt{do}\{y \leftarrow \mathtt{do}\{x \leftarrow s; t\}; u\} = \mathtt{do}\{x \leftarrow s; \mathtt{do}\{y \leftarrow t; u\}\}}$$

(To describe a particular notion of computation (I/O, mutable state, exceptions, concurrent processes, ...) one can consider extensions of vanilla CLC, e.g. with extra ground types, constants and equations.)

Parameterised Kleisli triple

is the following extra structure on a category $\mathbf{C}$ with binary products:

- a function mapping each $X \in \mathtt{obj}\,\mathbf{C}$ to an object $T(X) \in \mathtt{obj}\,\mathbf{C}$
- for each $X \in \mathtt{obj}\,\mathbf{C}$, a $\mathbf{C}$-morphism $X \xrightarrow{\eta_X} T(X)$
- for each $\mathbf{C}$-morphism $X \times Y \xrightarrow{f} T(Z)$ a $\mathbf{C}$-morphism $X \times T(Y) \xrightarrow{f^*} T(Z)$

satisfying...

Parameterised Kleisli triple[cont.]

- if $X \xrightarrow{f} X'$ and $X' \times Y \xrightarrow{g} T(Z)$, then

$$(g \circ (f \times \mathtt{id}_Y))^* = g^* \circ (f \times \mathtt{id}_{T(Y)})$$

- if $X \times Y \xrightarrow{f} T(Z)$, then

$$f^* \circ (\mathtt{id}_X \times \eta_Y) = f$$

- if $X \times Y \xrightarrow{f} T(Z)$ and $X \times Z \xrightarrow{g} T(W)$, then

$$(g^* \circ \langle \pi_1 , f \rangle)^* = g^* \circ \langle \pi_1 , f^* \rangle$$

Examples in **Set**

State: fix a set S (of "states") and define

$T(X) \triangleq (X \times S)^S$

$\eta_X\, x\, s \triangleq (x, s)$

$f^*(x, t)\, s \triangleq f(x, y)\, s'$ where $t\, s = (y, s')$

Examples in **Set**

State: fix a set S (of "states") and define

$$T(X) \triangleq (X \times S)^S$$

computations are functions $S \to X \times S$ taking states to values in X paired with a next state

$$\eta_X\, x\, s \triangleq (x, s)$$

$$f^*(x, t)\, s \triangleq f(x, y)\, s' \text{ where } t\, s = (y, s')$$

$f^*(x, \_)$ first "runs" $t \in T(Y)$ in state s to get (y, s'), then runs $f(x, y) \in T(Z)$ in the new state s'

Examples in **Set**

Error:

$$T(X) \triangleq X + 1 = \{(0, x) \mid x \in X\} \cup \{(1, 0)\}$$

$$\eta_X\, x \triangleq (0, x)$$

$$f^*(x, t) \triangleq \begin{cases} f(x, y) & \text{if } t = (0, y) \\ (1, 0) & \text{if } t = (1, 0) \end{cases}$$

Examples in **Set**

Error:

$$T(X) \triangleq X + 1 = \{(0,x) \mid x \in X\} \cup \{(1,0)\}$$

computations are either copies $(0,x)$ of values in $x \in X$ or an error $(1,0)$

$$\eta_X\, x \triangleq (0,x)$$

$$f^*(x,t) \triangleq \begin{cases} f(x,y) & \text{if } t = (0,y) \\ (1,0) & \text{if } t = (1,0) \end{cases}$$

if $t \in T(Y)$ is the error, then $f^*(x,\_)$ propagates it, otherwise it acts like f

Examples in **Set**

Continuations: fix a set R (of "results") and define

$$T(X) \triangleq R^{(R^X)}$$

$$\eta_X\, x \triangleq \lambda c \in R^X.\, c\, x$$

$$f^*(x, r) \triangleq \lambda c \in R^Z.\, r(\lambda y \in Y.\, f(x, y)\, c)$$

Examples in **Set**

Continuations: fix a set R (of "results") and define

$$T(X) \triangleq R^{(R^X)}$$

computations are functions $r : R^X \to R$ mapping continuations $c \in R^X$ of the computation to results $r\,c \in R$

$$\eta_X\, x \triangleq \lambda c \in R^X .\, c\, x$$

$$f^*(x, r) \triangleq \lambda c \in R^Z .\, r(\lambda y \in Y .\, f(x, y)\, c)$$

f^* maps a computation $r \in R^{(R^Y)}$ to the function taking a continuation $c \in R^Z$ to the result of applying r to the continuation $\lambda y \in Y .\, f(x, y)\, c$ in R^Y

Semantics of CLC

Given a ccc $\mathbf{C}$ equipped with a parameterised Kleisli triple $(T, \eta, (\_)^*)$, we can extend the semantics of STLC to one for CLC.

Computation types: $[\![\mathtt{T}(A)]\!] = T([\![A]\!])$

Trivial computations:

$$[\![\Gamma \vdash \mathtt{return}\, t : \mathtt{T}(A)]\!] = [\![\Gamma]\!] \xrightarrow{[\![\Gamma \vdash t:A]\!]} [\![A]\!] \xrightarrow{\eta_{[\![A]\!]}} T([\![A]\!])$$

Sequencing: $[\![\Gamma \vdash \mathtt{do}\{x \leftarrow s; t\} : \mathtt{T}(B)]\!] = f^* \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, g\rangle$

where $\begin{cases} f = [\![\Gamma]\!] \times [\![A]\!] \xrightarrow{[\![\Gamma, x:A \vdash t:\mathtt{T}(B)]\!]} T([\![B]\!]) \\ g = [\![\Gamma]\!] \xrightarrow{[\![\Gamma \vdash s:\mathtt{T}(A)]\!]} T([\![A]\!]) \end{cases}$

(and where A is uniquely determined from the proof of $\Gamma \vdash \mathtt{do}\{x \leftarrow s; t\} : \mathtt{T}(B)$)

Semantics of CLC

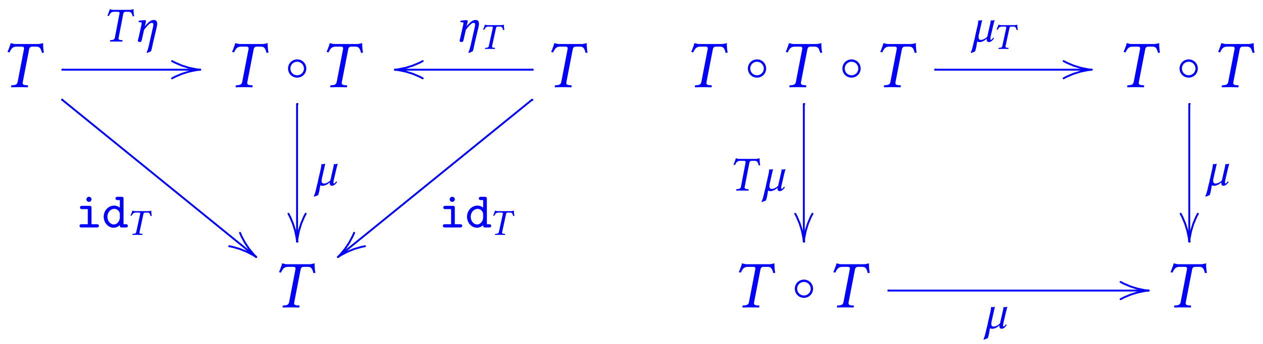
Given a ccc **C** equipped with a parameterised Kleisli triple $(T, \eta, (\_)^*)$, we can extend the semantics of STLC to one for CLC.

As for STLC versus cccs,

- the semantics of CLC in cc+Kleisli categories is equationally sound and complete
- one can use CLC as an internal language for describing constructs in cc+Kleisli categories
- there is a correspondence between equational theories in CLC and cc+Kleisli categories

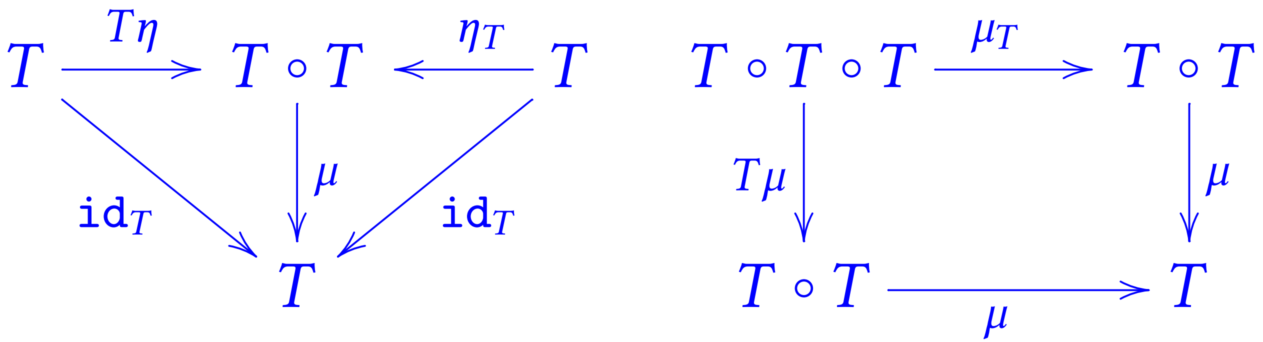
Monads

A monad on a category $\mathbf{C}$ is given by a functor $T : \mathbf{C} \to \mathbf{C}$ and natural transformations $\eta : \mathtt{id} \to T$ and $\mu : T \circ T \to T$ satisfying



Monads

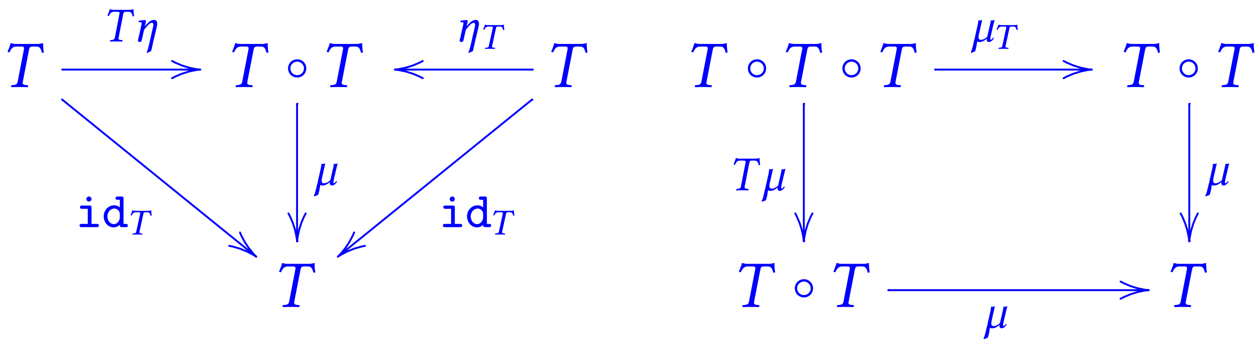
A monad on a category $\mathbf{C}$ is given by a functor $T : \mathbf{C} \to \mathbf{C}$ and natural transformations $\eta : \mathtt{id} \to T$ and $\mu : T \circ T \to T$ satisfying



If $\mathbf{C}$ has binary products, then the monad is strong if there is a family of $\mathbf{C}$-morphisms $(X \times T(Y) \xrightarrow{s_{X,Y}} T(X \times Y) \mid X, Y \in \mathtt{obj}\,\mathbf{C})$ satisfying a number (7, in fact) of commutative diagrams (details omitted, see Moggi).

Monads

A monad on a category $\mathbf{C}$ is given by a functor $T : \mathbf{C} \to \mathbf{C}$ and natural transformations $\eta : \mathtt{id} \to T$ and $\mu : T \circ T \to T$ satisfying



If $\mathbf{C}$ has binary products, then the monad is strong if there is a family of $\mathbf{C}$-morphisms $(X \times T(Y) \xrightarrow{s_{X,Y}} T(X \times Y) \mid X, Y \in \mathtt{obj}\,\mathbf{C})$ satisfying a number (7, in fact) of commutative diagrams (details omitted, see Moggi).

FACT: for a given category with binary products, "parameterised Kleisli triple" and "strong monad" are equivalent notions – each gives rise to the other in a bijective fashion.

Monads and adjunctions

- Given an adjunction $\mathbf{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathbf{D}$ $\quad F \dashv G$

  we get a monad $(G \circ F, \eta, \mu)$ on $\mathbf{C}$

  where $\begin{cases} \eta_X &= \overline{\mathtt{id}_{F\,X}} \\ \mu_X &= G(\overline{\mathtt{id}_{G(F\,X)}}) \end{cases}$

  E.g. for $\mathbf{Set} \underset{U}{\overset{F}{\rightleftarrows}} \mathbf{Mon}$ where U is the forgetful functor, $T = U \circ F$ is the list monad on $\mathbf{Set}$ ($T(X) = \mathtt{List}\,X$, η given by singleton lists, μ by flattening lists of lists). It's a strong monad (all monads of $\mathbf{Set}$ have a strength), but in general the monad associated with an adjunction may not be strong.

Monads and adjunctions

- Given an adjunction $\mathbf{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathbf{D}$ $\qquad$ $F \dashv G$

  we get a monad $(G \circ F, \eta, \mu)$ on $\mathbf{C}$

- Given a monad (T, η, μ) on $\mathbf{C}$ we get an adjunction

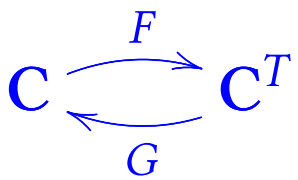
  $\mathbf{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathbf{C}^T$ $\qquad$ $F \dashv G$

Monads and adjunctions

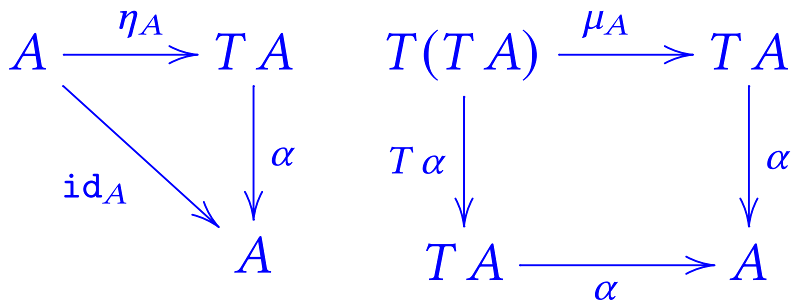
- Given an adjunct

  we get a monad (

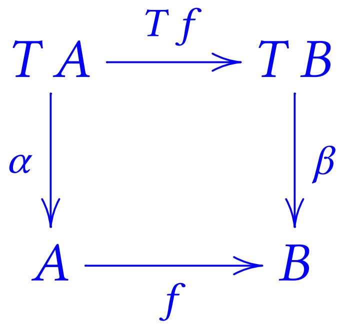
- Given a monad (



$\mathbf{C}^T$ is the category of Eilenberg-Moore algebras for the monad T, which has objects (A, α) with $\alpha : T(A) \longrightarrow A$ satisfying



and morphisms $f\,(A, \alpha) \longrightarrow (B, \beta)$ with $f : A \longrightarrow B$ satisfying



Monads and adjunctions

- Given an adjunction $\mathbf{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathbf{D}$ $\quad F \dashv G$

  we get a monad $(G \circ F, \eta, \mu)$ on $\mathbf{C}$

- Given a monad (T, η, μ) on $\mathbf{C}$ we get an adjunction

  $\mathbf{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathbf{C}^T$ $\quad F \dashv G$

- Starting from $\mathbf{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathbf{D}$ $\quad F \dashv G$ and forming the monad

  $T = G \circ F$, there's an obvious functor $K : \mathbf{D} \longrightarrow \mathbf{C}^T$.

  Monadicity Theorems impose conditions on $G : \mathbf{D} \to \mathbf{C}$ which ensure that K is an equivalence of categories. E.g. **Mon** is equivalent to the category of Eilenberg-Moore algebras for the list monad on **Set** (and similarly for any algebraic theory).

Some current themes involving category theory in computer science

- semantics of effects & co-effects in programming languages

  (monads and comonads)

- homotopy type theory

  (higher-dimensional category theory)

- structural aspects of networks, quantum computation/protocols, …

  (string diagrams for monoidal categories)