22/3/18.

Curry-Howard Seminar 2, Lecture 3.

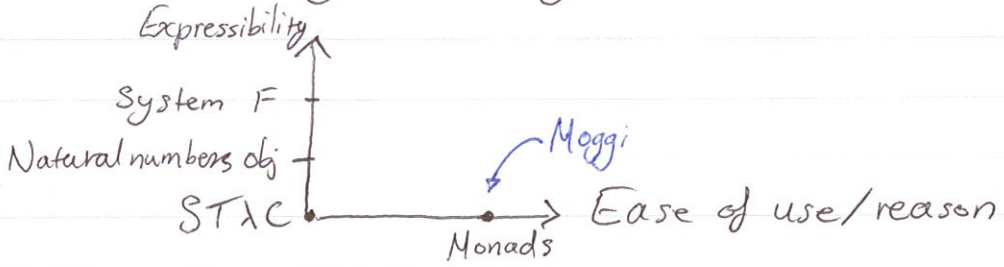
What are programs?

Easy! A program is just a term in ~~the~~ a simply typed λ-calculi.

Is it though...?

In fact, there are various reasons why terms in such a λ-calculus is not a sufficient notion of a program. As discussed in the previous lecture, STλC are quite weak, so one reason for extension is to increase how expressive the language is.

For example, this can be done by adding a natural numbers object to the corresponding Cartesian Closed Category, another way is to add a higher order structure, the System F gives such an example.



Somewhat orthogonal to this, is the concern of how easy to use and/or reason about programs. It was Moggi's suggestion that ~~Monads~~ a given ~~λ-calculi~~ λ-calculi be equipped with a monad to make it easier to reason about programs.

Moggi

$$\{\text{"Programs"}\} \cong \{\text{Cartesian Closed Categories + monads}\}$$

$\uparrow$ Lambek and Scott. $\downarrow$

$$\{\text{Simply typed } \lambda\text{-calculi}\} \cong \{\text{Cartesian Closed Categories}\}$$

21/3/18

Basic definitions:

Def^n 1:

A <u>Category</u> consists of

- A <u>class</u> of objects $\mathcal{C} = \{X, Y, Z, \dots\}$
- For each pair of objects (X, Y), a <u>set</u> of morphisms $\mathcal{C}(X, Y) = \{f : X \to Y, g : X \to Y, \dots\}$

Such that

- For every object $X \in \mathcal{C}$, there is an identity morphism $id_X : X \to X$
- For every triple of objects (X, Y, Z), a composition function

$$\circ : \mathcal{C}(X, Y) \times \mathcal{C}(Y, Z) \longrightarrow \mathcal{C}(X, Z)$$
$$(f, g) \longmapsto g \circ f$$

Satisfying

- $id_X \circ f = f \circ id_Y = f$
- $f \circ (g \circ h) = (f \circ g) \circ h$, whenever the compositions make sense.

Example 1)
Objects = Sets
Morphisms = Set functions.

Example 2) Let M be an arbitrary, fixed monoid (ie, a group without the requirements of inverse elements).
Objects = $\{*\}$ (the singleton set).
Morphisms: $m \in M \Rightarrow m: * \longrightarrow * \in M(*,*)$, where M is the Category of the monoid M of the same name.

Def^n 2: (Functor)
Let $\mathcal{C}$ and $\mathcal{D}$ be two Categories.
A <u>functor</u> $F: \mathcal{C} \longrightarrow \mathcal{D}$ assigns to each object $X \in \mathcal{C}$, an object $FX \in \mathcal{D}$, along with for each set $\mathcal{C}(X,Y)$, a function

$$F: \mathcal{C}(X,Y) \longrightarrow \mathcal{D}(FX, FY)$$

(where the notation F for this functor is used for convenience).

Satisfying
- $\forall X \in \mathcal{C}$, $F(id_X) = id_{FX}$
- $F(f \circ g) = Ff \circ Fg$, whenever composition makes sense.

Motivation:
This is done all the time. For example, in Algebraic Topology, want to study a top space by looking at an associated algebraic construction.

eg) Given a top space X and a point $x_0 \in X$, can construct the fundamental group $\pi_1(X, x_0)$, so

$$\pi_1 : \underline{Top_*} \longrightarrow \underline{Grp}$$

Category of pointed spaces.

is a functor.

Given two functors $F, G: \mathcal{C} \longrightarrow \mathcal{D}$, what is a map $\eta: F \Rightarrow G$?
What is the data of F & G?
Both are $\mathcal{C}$ indexed (family of) objects in $\mathcal{D}$
$\{FX\}_{X \in \mathcal{C}}$, $\{GX\}_{X \in \mathcal{C}}$
So a map $\eta: F \Rightarrow G$ should be a $\mathcal{C}$ indexed family of morphisms in $\mathcal{D}$, $\eta = \{\eta_X : FX \to GX\}_{X \in \mathcal{C}}$

but for any morphism $f: X \to Y \in \mathcal{C}(X,Y)$, there are two morphisms

$$FX \xrightarrow{Ff} FY$$

$$GX \xrightarrow[Gf]{} GY$$

so η should satisfy the condition that the diagram

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ {\scriptstyle \eta_X}\downarrow & \circlearrowright & \downarrow{\scriptstyle \eta_Y} \\ GX & \xrightarrow[Gf]{} & GY \end{array}$$

commutes, for all $f: X \to Y$.

$\mathcal{C}$ indexed
Any such family of morphisms in $\mathcal{D}$ satisfying this coherence condition is said to be a <u>natural transformation</u>. (Defn 3)

Example: Let M be a fixed monoid, and define the functor

$T: \underline{\mathbf{Set}} \longrightarrow \underline{\mathbf{Set}}$
$X \longmapsto X \times M$

Where for $f: X \to Y$,
$(Tf)(x,m) := (f(x), m)$.

Then define $\mu: T^2 \Rightarrow T$ by
$\mu_X : T^2(X) = (X \times M) \times M \to X \times M$
$((x, m_1), m_2) \mapsto (x, m_1 \cdot m_2)$

Monoid multiplication.

This is natural because for any $f: X \to Y$, and $((x, m_1), m_2) \in T^2(X)$,

$$\begin{aligned} (\mu_Y \circ T^2 f)((x, m_1), m_2) &= \mu_Y (Tf(x, m_1), m_2) \\ &= \mu_Y ((f(x), m_1), m_2) \\ &= (f(x), m_1 \cdot m_2) \\ &= Tf(x, m_1 \cdot m_2) \\ &= (Tf \circ \mu_X)((x, m_1), m_2) \end{aligned}$$

□

Examples:

- Let $T: Set \longrightarrow Set$

$$X \longmapsto \{(x_1, \dots, x_n) \mid n \in \mathbb{Z}_{\geq 0},\ x_i \in X\}$$

where for any morphism $f: X \longrightarrow Y$,

$$(Tf)(x_1, \dots, x_n) = (f(x_1), \dots, f(x_n)).$$

Then $\eta: id_{Set} \Rightarrow T$ given by,

$$\eta_X(x) = (x), \text{ for } X \in Set,\ x \in X,$$

is a natural transformation.

Pf: given $f: X \longrightarrow Y$, does

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle \eta_X}\downarrow & & \downarrow{\scriptstyle \eta_Y} \\ TX & \xrightarrow[Tf]{} & TY \end{array} \quad \text{commute?}$$

Yes! Because $\forall x \in X$,

$$(\eta_Y \circ f)(x) = \eta_Y(f(x)) = (f(x)) = (Tf)(x) = (Tf \circ \eta_X)(x). \quad \square$$

- $\mu: T^2 \Rightarrow T$ given by,

$$\mu_X((x_1^1, \dots, x_1^{k_1}), \dots, (x_n^1, \dots, x_n^{k_n})) = (x_1^1, \dots, x_1^{k_1}, \dots, x_n^1, \dots, x_n^{k_n})$$

is a natural transformation

Pf: given $f: X \to Y$,

$$\begin{array}{ccc} ((x_1^1, \dots, x_1^{k_1}), \dots, (x_n^1, \dots, x_n^{k_n})) & \longmapsto & ((f(x_1^1), \dots, f(x_1^{k_1})), \dots, ((f(x_n^1), \dots, f(x_n^{k_n})) \\ \downarrow & & \downarrow \\ T^2(X) & \xrightarrow{T^2(f)} & T^2(Y) \\ {\scriptstyle \mu_X}\downarrow & & \downarrow{\scriptstyle \mu_Y} \\ T(X) & \xrightarrow[T(f)]{} & T(Y) \\ \downarrow & & \downarrow \\ (x_1^1, \dots, x_1^{k_1}, \dots, x_n^1, \dots, x_n^{k_n}) & \longmapsto & (f(x_1^1), \dots, f(x_1^{k_1}), \dots, f(x_n^1), \dots, f(x_n^{k_n})) \end{array}$$

6/3/18.

The functor T along with the natural transformations η and μ give an example of a monad.

Def^n 4: (Monad).
A <u>monad</u> is a triple (T, η, μ) where
- $T: \mathcal{C} \to \mathcal{C}$ is a functor
- $\eta : id_{\mathcal{C}} \Rightarrow T$ and
- $\mu : T^2 \Rightarrow T$ are natural transformations;

Satisfying, $\forall X \in \mathcal{C}$, the diagrams

$$\begin{array}{ccc} T^3(X) & \xrightarrow{T\mu_X} & T^2(X) \\ {\scriptstyle \mu_{TX}}\downarrow & & \downarrow{\scriptstyle \mu_X} \\ T^2(X) & \xrightarrow[\mu_X]{} & T(X) \end{array} \quad \text{and} \quad \begin{array}{ccccc} T(X) & \xrightarrow{\eta_{TX}} & T^2(X) & \xleftarrow{T(\eta_X)} & T(X) \\ & {\scriptstyle id_{T(X)}}\searrow & \downarrow{\scriptstyle \mu_X} & \swarrow{\scriptstyle id_{T(X)}} & \\ & & T(X) & & \end{array}$$

commute.

In the special case when T: Set $\to$ Set, where $X \longmapsto M \times X$
for $f: X \to Y$, $T(f)(m,x) = (m, f(x))$, it can be shown that
(More precisely, there is a bijection $\{$Monoid structures on $M\} \cong \{$natural transformations μ, η, s.t (T, μ, η) is a monad$\}$)

M can be given a monoid structure
$\Leftrightarrow$
There exists μ, η such that (T, μ, η) is a monad.

So monads generalise monoids

In the $\Leftarrow$ direction of the proof, the monoid mult will be defined by μ and the unit by η, for this reason, in a general monad, μ is refered to as the <u>multiplication</u>, and η the <u>unit</u>.

Also in the $\Leftarrow$ direction of the proof, the square diagram will imply associativity of the multiplication, and the triangles one the unit requirement.

Great, but how do these help us to reason about programs?

It was Moggi's suggestion that programs ought not be identified with input/output pairs. A motivating example, is what if some program had some dependency on the state of the machine? Eg) a program which uses the current time during its computation, this program depends on more than just the given input.

Perhaps an obvious answer to this is to ~~just move the time~~ alter the ~~function so that~~ program as to make the time be given as an input, and then look at the input/output pairs of this new program. Indeed, this is exactly what monads do!

The remainder of this talk/article is dedicated to convincing the viewer of this.

This will be done in two stages. First, a definition will be given, followed by an explanation with examples of how this new object helps to reason about programs. Then, a proof will be given, showing that this new object is in fact equivalent to the notion of a monad.

Def^n 5: (Kleisli triple).
A Kleisli triple over a Category $\mathcal{C}$ is a triple $(T, \eta, \_^*)$, where,
- T: obj($\mathcal{C}$) $\to$ obj($\mathcal{C}$) is a function. (Not a functor).
- $\eta = \{\eta_C : C \to TC\}_{C \in obj(\mathcal{C})}$ a collection of functions. (Not a nat trans).
- for each function $f: A \to TB$, a function $f^*: TA \to TB$

Satisfying:
- $\eta^*_{TC} = id_{TC}$ $\forall C \in$ obj($\mathcal{C}$).
- $f^* \circ \eta_C = f$, for all $f: C \to TB$.
- $(g^* \circ f^*) = (g^* \circ f)^*$, whenever composition makes sense.

Example.
Say a program might fail to return an output on some given input, then this can be modelled as a function which returns $\perp$, a formal symbol representing failure, on some inputs.
Formally,

Let . $TA = A \amalg \{\perp\}$
. $\eta_A : A \to A \amalg \{\perp\}$ canonical inclusion
. given $f : A \to TB$, define

$$f^*(x) = \begin{cases} f(x), & x \in A \\ \perp, & x = \perp \end{cases}$$

given $g : A \to TB$ & $f : B \to TC$, want to form $f \cdot g$, but this is ill defined.

Instead, let composition be $f^* \circ g$.
f^* "passes on the failure".

A more involved example:
What if a function has an output of type A, but this output depends on the state of the machine, and the computation f alters the state.

Dependence on state can be thought of as a function from a set of states S to A, and the alteration of state can be modelled by returning what state f altered the machine to.

So $TA = (A \times S)^S$, and for $f : A \to TB$,

$$f^* : (A \times S)^S \to (B \times S)^S$$
$$g \mapsto (f \circ \pi_1 \circ g, \pi_2 \circ g)$$

How are Kleisli triples "the same" as monads? This comes from the following theorem,

Thm 6:
There is a bijection between Kleisli triples and monads.

The bijection is given by:

Given a kleisli triple $(T, \eta, \_^*)$, extend T to morphisms by $T(f: A \to B) = (\eta_B f)^*$, and

For any $X \in \mathcal{C}$, let $\mu_X = id_{TX}^*$

~~Naturality of μ follows from the fact that~~

Conversely, given a monad (T, η, μ), for any given $f: A \to TB$, let $f^* = \mu_B(Tf)$.

The naturality a axioms of a monad correspond to the axioms for a kleisli triple.

In fact, monads aren't the most fundamental objects here, indeed every pair of adjoint functors gives rise to a monad. Moreover, every monad gives rise to an adjoint pair of functors, this is proven by constructing a category based on the monad called the kleisli category.

Also, monads aren't a perfect answer. For example, what, if one wished to discuss a program which both depended on state and could possibly fail?

Say T and S are monads, then what is μ^{TS} for TS?

$$\forall C \in \mathcal{C},\ \mu^{TS}_C : TSTS(c) \longrightarrow TS(c)$$

We have $\mu^T : T^2 \longrightarrow T$ and
$\mu^S : S^2 \longrightarrow S$, so need

$ST \longrightarrow TS$, but there is no canonical way of doing this in general.

So monads dont compose.