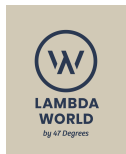




Emily Riehl

Johns Hopkins University

# A categorical view of computational effects



Lambda World Cádiz



Let  $\mathbb{T}$  denote a computational effect.

- A  $\mathbb{T}$ -program is a function  $A \xrightarrow{f} \mathbb{T}(B)$  from the set of values of type  $A$  to the set of  $\mathbb{T}$ -computations of type  $B$ .
- $\mathbb{T}$  is a **monad** just when it has the structure needed to turn  $\mathbb{T}$ -programs into a category.
- The  $\mathbb{T}$ -programs between finite types define a **Lawvere theory**.
- The Lawvere theory presents the operations and equations for the computational effect  $\mathbb{T}$ .\*

\*If  $\mathbb{T}$  is not **finitary**, these operations and equations define a different monad.



0. Functions, composition, and categories
1. Categories for computational effects (monads)
2. Categories of operations and equations (Lawvere theories)
3. Lawvere theories = (finitary) monads



# Functions, composition, and categories

# The mathematician's view of functions



A function, e.g.:

$$f(x) = x^2 - x$$

always comes with specified sets of “possible input values” and “potential output values.” One writes

$$I \xrightarrow{f} O$$

to indicate that  $f$  is a function with **source**  $I$  and **target**  $O$ .

**Why bother with sources and targets?** This data indicates when two functions are **composable**:

$$A \xrightarrow{f} B \quad \text{and} \quad B \xrightarrow{g} C$$

are **composable** just when the target of  $f$  equals the source of  $g$ .

# What is a category?

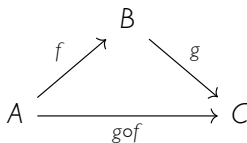


A **category** is a two-sorted structure that encodes the **algebra of composition**. It has

- **objects**:  $A, B, C \dots$  and
- **arrows**:  $A \xrightarrow{f} B, B \xrightarrow{g} C$ , each with a specified source and target

so that

- each pair of **composable arrows**:



has a **composite arrow**

- and each object has an **identity arrow**  $A \xrightarrow{\text{id}_A} A$

for which the composition operation is **associative** and **unital**.

# What is the point of identity arrows?



An **isomorphism** consists of:

$$A \begin{array}{c} \xrightarrow{f} \\ \xleftarrow{g} \end{array} B$$

so that

$$g \circ f = \text{id}_A \quad \text{and} \quad f \circ g = \text{id}_B$$

**Isomorphism invariance principle:**  
If  $A$  and  $B$  are isomorphic then every category theoretic property of  $A$  is also true of  $B$ .

# Examples of categories



In the category **Set** the

- **objects** are (finite) sets  $X, Y, \dots$
- **arrows** are functions  $X \xrightarrow{f} Y, \dots$

In the **syntactic category** for some programming language the

- **objects** are types  $X, Y, \dots$
- **arrows** are programs  $X \xrightarrow{f} Y, \dots$

Note that the same notation describes the data in any category. The precise ontology of the objects and arrows won't matter much.





# Categories for computational effects (monads)



Let us introduce some constructions

$$\text{Set} \xrightarrow{\mathbb{T}} \text{Set}$$

each encoding a computational effect:

- $\text{list}(X) :=$  finite lists of elements of  $X$
- $\text{maybe}(X) := X + \{\perp\}$
- $\text{exceptions}_E(X) := X + E$
- $\text{side-effects}_S(X) := \{S \rightarrow S \times X\}$
- $\text{non-det}(X) := \{\text{finite non-empty subsets of } X\}$
- $\text{prob-dist}(X) := \{X \xrightarrow{p} [0, 1] \mid \sum_{x \in X} p(x) = 1\}$
- $\text{continuations}_R(X) := \{(X \rightarrow R) \rightarrow R\}$



For any notion of computation  $\mathbb{T}$

- $\text{list}(X) :=$  finite lists of elements of  $X$
- $\text{maybe}(X) := X + \{\perp\}$
- $\text{exceptions}_E(X) := X + E$
- $\text{side-effects}_S(X) := \{S \rightarrow S \times X\}$
- $\text{non-det}(X) := \{\text{finite non-empty subsets of } X\}$
- $\text{prob-dist}(X) := \{X \xrightarrow{p} [0, 1] \mid \sum_{x \in X} p(x) = 1\}$
- $\text{continuations}_R(X) := \{(X \rightarrow R) \rightarrow R\}$

a  $\mathbb{T}$ -program from  $A$  to  $B$  is a function  $A \xrightarrow{f} \mathbb{T}(B)$ , from the set of values of type  $A$  to the set of  $\mathbb{T}$ -computations of type  $B$ .

Write  $A \xrightarrow{f} B$  to mean  $A \xrightarrow{f} \mathbb{T}(B)$ .

# Programs should form a category



A  $\mathbb{T}$ -program from  $A$  to  $B$  is a function  $A \xrightarrow{f} \mathbb{T}(B)$ , from the set of values of type  $A$  to the set of  $\mathbb{T}$ -computations of type  $B$ .

The notion of **monad** arises from the following categorical imperative:

programs should form a category

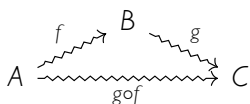
**Slogan:** A computational effect  $\mathbb{T}$  defines a **monad** just when the  $\mathbb{T}$ -programs  $A \xrightarrow{f} B$  define the arrows in a **category** of  $\mathbb{T}$ -programs.

# The category of $\mathbb{T}$ -programs



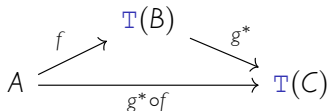
To define the **category of  $\mathbb{T}$ -programs**  $\mathbf{Kl}_{\mathbb{T}}$  we need:

- identity arrows  $A \xrightarrow{\text{id}_A} A$ ; a monad has **pure functions**  $A \xrightarrow{\text{pure}} \mathbb{T}(A)$
- a composition rule for  **$\mathbb{T}$ -computations**:



**Problem:**  $A \xrightarrow{f} \mathbb{T}(B)$  and  $B \xrightarrow{g} \mathbb{T}(C)$  are not composable!

With a monad, any function  $B \xrightarrow{g} \mathbb{T}(C)$  can be extended to a function  $\mathbb{T}(B) \xrightarrow{g^*} \mathbb{T}(C)$  via the **bind operation**. Then



defines the **Kleisli composite** of  $A \xrightarrow{f} B$  and  $B \xrightarrow{g} C$ .

# The category $\mathbf{Kl}_{\text{maybe}}$ of maybe-computations



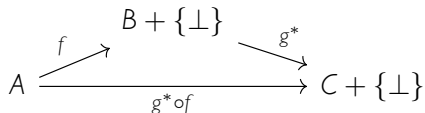
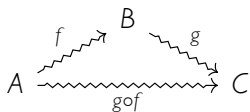
For  $\text{maybe}(X) := X + \{\perp\}$

- A **maybe**-program  $A \xrightarrow{f} B$  is a function  $A \xrightarrow{f} B + \{\perp\}$ , i.e., a **partial function** from  $A$  to  $B$ .
- The identity  $A \xrightarrow{\text{id}_A} A$  is the function  $A \xrightarrow{\text{incl}} A + \{\perp\}$ .
- Any function  $B \xrightarrow{g} C + \{\perp\}$  extends to a function

$$B + \{\perp\} \xrightarrow{g^*} C + \{\perp\}$$

by the rule  $g^*(\perp) = \perp$ .

- The Kleisli composite



is the **largest partial function** from  $A$  to  $C$ .

# The category $\mathbf{Kl}_{\text{list}}$ of list-computations



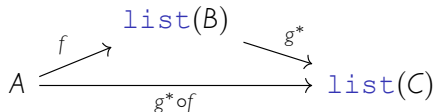
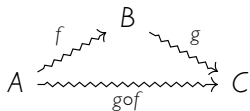
For  $\text{list}(X)$

- A  $\text{list}$ -program  $A \xrightarrow{f} B$  is a function  $A \xrightarrow{f} \text{list}(B)$ , i.e., a function from  $A$  to lists in  $B$ .
- The identity  $A \xrightarrow{\text{id}_A} \text{list}(A)$  is the function  $A \xrightarrow{\text{singleton}} \text{list}(A)$ .
- Any function  $B \xrightarrow{g} \text{list}(C)$  extends to a function

$$\text{list}(B) \xrightarrow{g^*} \text{list}(C)$$

by applying  $g$  to each term in a list of elements of  $B$  and concatenating the result.

- The Kleisli composite



is defined by application of  $f$  and  $g$  followed by **concatenation**.



2

Categories of operations and equations (Lawvere theories)



# Kleisli arrows define operations



Let  $\underline{n} := \{x_1, \dots, x_n\}$  denote the set with  $n$  elements.

An arrow  $\underline{1} \rightsquigarrow \underline{n}$  in the category of `list`-programs  $\mathbf{Kl}_{\text{list}}$  is

- a function  $\underline{1} \rightarrow \text{list}(\underline{n})$  (by definition) or equivalently
- an element of  $\text{list}(\underline{n})$  (the image of the previous function).

E.g.

$$\underline{1} \xrightarrow{x_3x_5x_2x_5} \underline{6} \quad \longleftrightarrow \quad x_3x_5x_2x_5 \in \text{list}(\underline{6})$$

which encodes a 6-ary operation “ $\lambda.x_3x_5x_2x_5.$ ”

Arrows  $\underline{1} \rightsquigarrow \underline{n}$  define  $n$ -ary operations.

# Kleisli composites define equations between operations



Arrows  $\underline{1} \rightsquigarrow \underline{n}$  in the category of `list`-programs  $\mathbf{Kl}_{\text{list}}$  define  $n$ -ary operations.

Compositions  $\begin{array}{ccc} & \underline{n} & \\ \nearrow & & \searrow \\ \underline{1} & \rightsquigarrow & \underline{m} \end{array}$  define equations between operations.

E.g.

$$\begin{array}{ccc} & \underline{2} & \\ x_1x_2 \rightsquigarrow & & (x_1, x_2x_3) \\ \underline{1} \rightsquigarrow & \rightsquigarrow & \underline{3} \\ & x_1x_2x_3 & \end{array} \quad \text{corresponds to} \quad x_1(x_2x_3) = x_1x_2x_3.$$

$$\begin{array}{ccc} & \underline{2} & \\ x_2x_1 \rightsquigarrow & & (x_3, x_1x_2) \\ \underline{1} \rightsquigarrow & \rightsquigarrow & \underline{3} \\ & x_1x_2x_3 & \end{array} \quad \text{corresponds to} \quad (x_1x_2)x_3 = x_1x_2x_3.$$

Together these operations and equations define the `list`-theory  $\mathbf{L}_{\text{list}}$ .

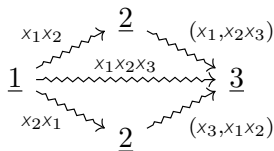
# Lawvere theories from monads



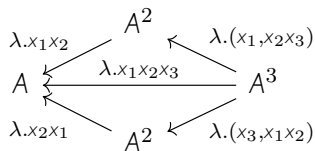
A **model** for the `list`-theory  $L_{\text{list}}$  is:

- a set  $A$
- together with a function  $A^n \rightarrow A$  for each  $n$ -ary operation  $\underline{1} \rightsquigarrow \underline{n}$
- satisfying the equations determined by the compositions in the category of `list`-programs.

E.g.



are modeled by



For any monadic computational effect  $T$ , let  $L_T$  denote the category of  $T$ -programs between finite sets. The opposite category  $L_T^{\text{op}}$ , obtained by formally reversing the arrows, defines a **Lawvere theory**.



3

Lawvere theories = (finitary) monads

# monads vs Lawvere theories



A monad is

- a “computational effect”  $\text{Set} \xrightarrow{\mathbb{T}} \text{Set}$
- so that  $\mathbb{T}$ -programs  $A \xrightarrow{f} \mathbb{T}(B)$  define the arrows  $A \xrightarrow{\tilde{f}} B$  in a category  $\text{Kl}_{\mathbb{T}}$ .

The opposite of the category of  $\mathbb{T}$ -programs between finite sets defines a Lawvere theory  $L_{\mathbb{T}}^{\text{op}}$ . Conversely, any Lawvere theory  $L$  defines a monad  $\mathbb{T}_L$  on  $\text{Set}$ .

**Theorem:** The category of Lawvere theories is equivalent to the category of finitary monads\* on  $\text{Set}$ .

Finitary monads and Lawvere theories describe equivalent categorical encodings of universal algebra.

# Advantages of Lawvere theories



Why bother with Lawvere theories if they are equivalent to monads?

- Each monad acts on just one category, whereas **models of Lawvere theories can be defined in any category** with finite products — and the construction of the category of models is functorial in both arguments.
- **Lawvere theory operations can be added:** any two Lawvere theories  $L$  and  $L'$  have a **sum**  $L + L'$  — indeed the category of Lawvere theories is locally finitely presentable.
- **Lawvere theory operations can be intertwined:** any two Lawvere theories  $L$  and  $L'$  have a **tensor product**  $L \otimes L'$ .
- In practice, **Lawvere theories are generated by computationally natural operations satisfying computationally meaningful equations** — e.g., exceptions, side-effects, interactive input-output, binary non-determinism, probabilistic non-determinism ...



All of computational effects mentioned thusfar fit into this framework for categorical universal algebra with one exception:

Even for  $\underline{2} = \{\top, \perp\}$ , the continuations monad

$$\text{continuations}_{\underline{2}}(X) := \{(X \rightarrow \underline{2}) \rightarrow \underline{2}\} = \mathfrak{P}(\mathfrak{P}(X))$$

is not finitary. It does define a [large](#) Lawvere theory, but this is specified with a proper class of operations.

“It appears that the continuations monad transformer should be seen as something sui generis.”

– Martin Hyland and John Power



Let  $\mathbb{T}$  denote a computational effect.

- A  $\mathbb{T}$ -program is a function  $A \xrightarrow{f} \mathbb{T}(B)$  from the set of values of type  $A$  to the set of  $\mathbb{T}$ -computations of type  $B$ .
- $\mathbb{T}$  is a **monad** just when it has the structure needed to turn  $\mathbb{T}$ -programs into a category.
- The  $\mathbb{T}$ -programs between finite types define a **Lawvere theory**.
- The Lawvere theory presents the operations and equations for the computational effect  $\mathbb{T}$ .\*

\*If  $\mathbb{T}$  is not **finitary**, these operations and equations define a different monad.



# References



- Eugenio Moggi, “Computational lambda-calculus and monads”  
— describes monads and the category of programs
- Gordon Plotkin and John Power, “Computational Effects and Operations: An Overview”  
— describes the connection between Moggi’s monads and Lawvere theories
- Martin Hyland and John Power, “The Category Theoretic Understanding of Universal Algebra: Lawvere Theories and Monads”  
— inspired this talk

Gracias!