Emily Riehl

Johns Hopkins University

A categorical view of computational effects

Lambda World Cádiz

Preview

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

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

*If $\mathtt{T}$ 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

0

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:

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

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

so that

- each pair of composable arrows:

$$\begin{tikzcd} & B \arrow[dr, "g"] & \\ A \arrow[ur, "f"] \arrow[rr, "g\circ f"'] & & C \end{tikzcd}$$

  has a composite arrow

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

for which the composition operation is associative and unital.

What is the point of identity arrows?

An isomorphism consists of:

$$A \underset{g}{\overset{f}{\rightleftarrows}} B$$

so that

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

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 $X, Y, \ldots$
- arrows are functions $X \xrightarrow{f} Y, \ldots$

In the syntactic category for some programming language the

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

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

1

Categories for computational effects (monads)

Computational effects

Let us introduce some constructions

$$\mathsf{Set} \xrightarrow{\mathtt{T}} \mathsf{Set}$$

each encoding a computational effect:

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

$\mathtt{T}$-programs

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

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

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

$$\text{Write} \qquad A \overset{f}{\rightsquigarrow} B \qquad \text{to mean} \qquad A \xrightarrow{f} \mathtt{T}(B).$$

Programs should form a category

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

The notion of monad arises from the following categorical imperative:

programs should form a category

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

The category of $\mathtt{T}$-programs

To define the category of $\mathtt{T}$-programs $\mathsf{Kl}_{\mathtt{T}}$ we need:

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

$$\begin{tikzcd} & B \arrow[dr, rightsquigarrow, "g"] & \\ A \arrow[ur, rightsquigarrow, "f"] \arrow[rr, rightsquigarrow, "g \circ f"'] & & C \end{tikzcd}$$

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

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

$$\begin{tikzcd} & \mathtt{T}(B) \arrow[dr, "g^*"] & \\ A \arrow[ur, "f"] \arrow[rr, "g^* \circ f"'] & & \mathtt{T}(C) \end{tikzcd}$$

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

The category $\mathsf{Kl}_{\texttt{maybe}}$ of `maybe`-computations

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

- A `maybe`-program $A \overset{f}{\rightsquigarrow} B$ is a function $A \xrightarrow{f} B + \{\perp\}$, i.e., a partial function from A to B.
- The identity $A \overset{\mathrm{id}_A}{\rightsquigarrow} A$ is the function $A \xrightarrow{\mathrm{incl}} A + \{\perp\}$.
- 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

$$\begin{tikzcd} & B \arrow[dr, rightsquigarrow, "g"] & \\ A \arrow[ur, rightsquigarrow, "f"] \arrow[rr, rightsquigarrow, "g \circ f"'] & & C \end{tikzcd} \qquad \begin{tikzcd} & B + \{\perp\} \arrow[dr, "g^*"] & \\ A \arrow[ur, "f"] \arrow[rr, "g^* \circ f"'] & & C + \{\perp\} \end{tikzcd}$$

is the largest partial function from A to C.

The category $\mathsf{Kl}_{\texttt{list}}$ of $\texttt{list}$-computations

For $\texttt{list}(X)$

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

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

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

- The Kleisli composite

$$\begin{tikzcd} & B \arrow[dr, squiggly, "g"] & \\ A \arrow[ur, squiggly, "f"] \arrow[rr, squiggly, "g \circ f"'] & & C \end{tikzcd} \qquad \begin{tikzcd} & \texttt{list}(B) \arrow[dr, "g^*"] & \\ A \arrow[ur, "f"] \arrow[rr, "g^* \circ f"'] & & \texttt{list}(C) \end{tikzcd}$$

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

2

Categories of operations and equations (Lawvere theories)

Kleisli arrows define operations

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

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

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

E.g.

$$\underline{1} \xrightarrow{x_3 x_5 x_2 x_5} \underline{6} \qquad \leftrightsquigarrow \qquad x_3 x_5 x_2 x_5 \in \mathtt{list}(\underline{6})$$

which encodes a 6-ary operation "$\lambda . x_3 x_5 x_2 x_5$."

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 $\mathrm{Kl}_{\texttt{list}}$ define n-ary operations.

Compositions
$$\begin{tikzcd} & \underline{n} \arrow[dr, squiggly] & \\ \underline{1} \arrow[ur, squiggly] \arrow[rr, squiggly] & & \underline{m} \end{tikzcd}$$
define equations between operations.

E.g.

$$\begin{tikzcd} & \underline{2} \arrow[dr, squiggly, "{(x_1, x_2x_3)}"] & \\ \underline{1} \arrow[ur, squiggly, "x_1x_2"] \arrow[rr, squiggly, "x_1x_2x_3"'] & & \underline{3} \end{tikzcd}$$
corresponds to $x_1(x_2x_3) = x_1x_2x_3.$

$$\begin{tikzcd} & \underline{2} \arrow[dr, squiggly, "{(x_3, x_1x_2)}"] & \\ \underline{1} \arrow[ur, squiggly, "x_2x_1"] \arrow[rr, squiggly, "x_1x_2x_3"'] & & \underline{3} \end{tikzcd}$$
corresponds to $(x_1x_2)x_3 = x_1x_2x_3.$

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

Lawvere theories from monads

A model for the `list`-theory $\mathsf{L}_{\texttt{list}}$ is:

- a set A
- together with a function $A^n \to A$ 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.

$$
\begin{tikzcd}
 & \underline{2} \arrow[dr, squiggly, "{(x_1,x_2x_3)}"] & \\
\underline{1} \arrow[ur, squiggly, "x_1x_2"] \arrow[rr, squiggly, "x_1x_2x_3"] \arrow[dr, squiggly, "x_2x_1"'] & & \underline{3} \\
 & \underline{2} \arrow[ur, squiggly, "{(x_3,x_1x_2)}"'] &
\end{tikzcd}
\quad \text{are modeled by} \quad
\begin{tikzcd}
 & A^2 \arrow[dl, "\lambda.x_1x_2"'] & \\
A & & A^3 \arrow[ll, "\lambda.x_1x_2x_3"'] \arrow[ul, "{\lambda.(x_1,x_2x_3)}"'] \arrow[dl, "{\lambda.(x_3,x_1x_2)}"] \\
 & A^2 \arrow[ul, "\lambda.x_2x_1"] &
\end{tikzcd}
$$

For any monadic computational effect $\texttt{T}$, let $\mathsf{L}_{\texttt{T}}$ denote the category of $\texttt{T}$-programs between finite sets. The opposite category ${\mathsf{L}_{\texttt{T}}}^{\mathsf{op}}$, 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" Set $\xrightarrow{\mathtt{T}}$ Set
- so that $\mathtt{T}$-programs $A \xrightarrow{f} \mathtt{T}(B)$ define the arrows $A \overset{f}{\rightsquigarrow} B$ in a category $\mathsf{Kl}_{\mathtt{T}}$.

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

Theorem: The category of Lawvere theories is equivalent to the category of finitary monads* on 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 $\mathsf{L} + \mathsf{L}'$— 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 $\mathsf{L} \otimes \mathsf{L}'$.
- 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 …

Continuations

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

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

$$\texttt{continuations}_{\underline{2}}(X) := \{(X \to \underline{2}) \to \underline{2}\} = \mathfrak{P}(\mathfrak{P}(X))$$

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

Review

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

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

*If $\mathtt{T}$ 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!