Emily Riehl

Johns Hopkins University

A categorical view of computational effects

Compose::Conference

1. Functions, composition, and categories

2. Categories for computational effects (monads)

3. Categories of operations and equations (Lawvere theories)

4. Lawvere theories vs monads

Preview

Let $\mathtt{T}$ denote a notion of computation.

- 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.

1

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.

Composable and non-composable functions

Are the functions

$$f(x) = x^2 - x \qquad \text{and} \qquad g(y) = \left(\frac{1}{2}\right)^y$$

composable? It depends. If

$$\mathbb{N} \xrightarrow{f} \mathbb{Z} \qquad \text{and} \qquad \mathbb{Z} \xrightarrow{g} \mathbb{Q}$$

where $\mathbb{N}$ is the set of natural numbers, $\mathbb{Z}$ is the set of integers, and $\mathbb{Q}$ is the set of rational numbers then yes: $(g \circ f)(x) = \left(\frac{1}{2}\right)^{x^2 - x}$. But if

$$\mathrm{Mat}_{2\times 2}(\mathbb{Z}) \xrightarrow{f} \mathrm{Mat}_{2\times 2}(\mathbb{Z}) \qquad \text{and} \qquad \mathbb{Z} \xrightarrow{g} \mathbb{Q}$$

where $\mathrm{Mat}_{2\times 2}(\mathbb{Z})$ is the set of 2×2-matrices with integer coefficients then no: what is the meaning of $\left(\frac{1}{2}\right)^y$ if y is a matrix?

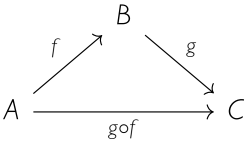
What is a category?

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

- for any pair of composable arrows:



  there exists 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 identities?

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.

2

Categories for computational effects (monads)

Notions of computation

Let us introduce some large functions

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

each encoding some notion of computation:

- $\mathtt{list}(X)$:= finite lists of elements of X
- $\mathtt{partial}(X) := X + \{\bot\}$
- $\mathtt{side\text{-}effects}_S(X) := [S, S \times X]$, the set of functions from S to $S \times X$
- $\mathtt{continuations}_R(X) := [[X, R], R]$, the set of functions from the set $[X, R]$ of functions from X to R to R
- $\mathtt{non\text{-}det}(X) := P_+(X)$, the set of non-empty subsets of X
- $\mathtt{prob\text{-}dist}(X)$:= the set of probability functions $X \xrightarrow{p} [0, 1]$ so that $\sum_{x \in X} p(x) = 1$

$\mathtt{T}$-programs

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

- $\mathtt{list}(X) :=$ finite lists of elements of X
- $\mathtt{partial}(X) := X + \{\bot\}$
- $\mathtt{side\text{-}effects}_S(X) := [S, S \times X]$, the set of functions from S to $S \times X$
- $\mathtt{continuations}_R(X) := [[X, R], R]$, the set of functions from the set $[X, R]$ of functions from X to R to R
- $\mathtt{non\text{-}det}(X) := P(X)$, the set of all subsets of X
- $\mathtt{prob\text{-}dist}(X) :=$ the set of probability functions $X \xrightarrow{p} [0, 1]$ so that $\sum_{x \in X} p(x) = 1$

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. Write

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

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

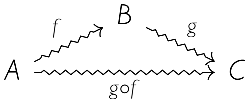
Theorem: A notion of computation $\mathtt{T}$ defines a monad if* and only if the $\mathtt{T}$-programs $A \overset{f}{\rightsquigarrow} B$ define the arrows in a category.

*If the category of $\mathtt{T}$-programs is constructed using a Kleisli triple, as depicted on the next slide, then $\mathtt{T}$ defines a monad.

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

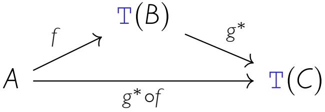
To define the category of $\mathtt{T}$-programs we need:

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



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 $\mathtt{T}(B) \xrightarrow{g^*} \mathtt{T}(C)$. Then



defines the Kleisli composite of $A \overset{f}{\rightsquigarrow} B$ and $B \overset{g}{\rightsquigarrow} C$ in the category $\mathrm{Kl}_{\mathtt{T}}$.

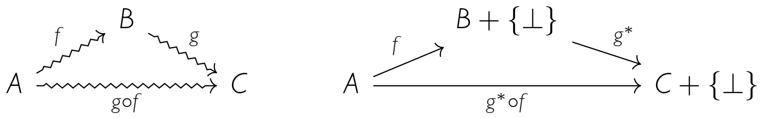
The category of `partial`-computations

For $\texttt{partial}(X) := X + \{\bot\}$

- A program $A \overset{f}{\rightsquigarrow} B$ is a function $A \xrightarrow{f} B + \{\bot\}$, i.e., a partial function from A to B.
- The unit $A \overset{\mathrm{id}_A}{\rightsquigarrow} A$ is the function $A \xrightarrow{\mathrm{incl}} A + \{\bot\}$.
- Any function $B \xrightarrow{g} C + \{\bot\}$ extends to a function

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

  by the rule $g^*(\bot) = \bot$.
- The Kleisli composite



  is the largest partial function from A to C.

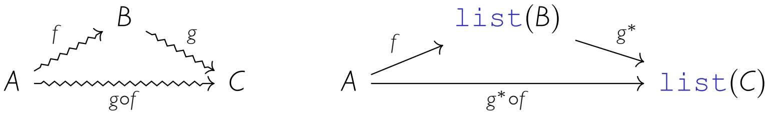
The category of `list`-computations

For $\texttt{list}(X)$

- A 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 unit $A \overset{\mathrm{id}_A}{\rightsquigarrow} \texttt{list}(A)$ 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



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

3

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 $\texttt{list}$-programs $\mathsf{Kl}_{\texttt{list}}$ is

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

E.g. $\underline{1} \overset{x_3x_5x_2x_5}{\rightsquigarrow} \underline{6}$ $\quad x_3x_5x_2x_5 \in \texttt{list}(\underline{6})$

Arrows $\underline{1} \rightsquigarrow \underline{n}$, i.e., elements of $\texttt{list}(\underline{n})$, define n-ary operations.

Kleisli composites define equations between operations

In the category of `list`-programs $\mathsf{Kl}_{\texttt{list}}$, arrows $\underline{1} \rightsquigarrow \underline{n}$ define n-ary operations.

> Compositions $\underline{1} \rightsquigarrow \underline{n} \rightsquigarrow \underline{m}$, $\underline{1} \rightsquigarrow \underline{m}$ define equations between operations.

E.g.

$$\underline{1} \overset{x_1x_2}{\rightsquigarrow} \underline{2} \overset{(x_1, x_2x_3)}{\rightsquigarrow} \underline{3}, \qquad \underline{1} \overset{x_1x_2x_3}{\rightsquigarrow} \underline{3}$$

corresponds to the equation

$$x_1(x_2x_3) = x_1x_2x_3.$$

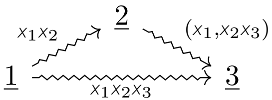
Together these operations and equations define the `list`-theory $\mathsf{L}_{\texttt{list}}$.

Models for the list-theory

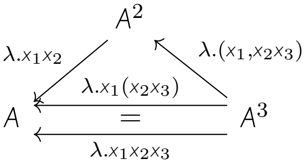
A model for the list-theory $\mathsf{L}_{\texttt{list}}$ is:

- 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 programs.

E.g.



are modeled by



A model for the list-theory $\mathsf{L}_{\texttt{list}}$ is a contravariant product-preserving functor from the category $\mathsf{L}_{\texttt{list}}$ of list-programs between finite sets.

4

Lawvere theories vs monads

models for a Lawvere theory

If $\mathtt{T}$ is any monadic notion of computation let $\mathsf{L}_{\mathtt{T}}$ denote the category of $\mathtt{T}$-programs between finite sets. The opposite category ${\mathsf{L}_{\mathtt{T}}}^{\mathsf{op}}$, obtained by formally reversing the arrows, defines a Lawvere theory.

A model is a functor $\mathsf{L}_{\mathtt{T}}^{\mathsf{op}} \to \mathsf{Set}$ defined on objects by

$$
\begin{array}{ccc}
\underline{1} & \longmapsto & A \\
\underline{2} & \longmapsto & A^2 \\
\vdots & & \vdots \\
\underline{n} & \longmapsto & A^n
\end{array}
$$

and carrying each arrow $\underline{n} \rightsquigarrow \underline{m}$ in the category $\mathsf{L}_{\mathtt{T}}$ to a function $A^m \to A^n$.

algebras for a monad

If $\mathtt{T}$ is any monadic notion of computation an algebra for $\mathtt{T}$ is a set A together with a function $\mathtt{T}(A) \xrightarrow{\alpha} A$ so that the composition relations hold:

$$
\begin{array}{ccc}
A & \xrightarrow{\eta_A} & \mathtt{T}(A) \\
& {\scriptstyle \mathrm{id}_A} \searrow & \downarrow \alpha \\
& & A
\end{array}
\qquad\qquad
\begin{array}{ccc}
\mathtt{T}(\mathtt{T}(A)) & \xrightarrow{\mathrm{id}^*_{\mathtt{T}(A)}} & \mathtt{T}A \\
\mathtt{T}\alpha \downarrow & & \downarrow \alpha \\
\mathtt{T}A & \xrightarrow[\alpha]{} & A
\end{array}
$$

Theorem: The category of models for the Lawvere theory ${\mathsf{L}_{\mathtt{T}}}^{\mathrm{op}}$ and the category of algebra for the monad $\mathtt{T}$ are equivalent.

monads vs Lawvere theories

A monad is

- a "notion of composition" Set $\xrightarrow{T}$ Set
- so that $\mathtt{T}$-programs $A \xrightarrow{f} \mathtt{T}(B)$ define the arrows $A \overset{f}{\rightsquigarrow} B$ in a category $\mathrm{Kl}_{\mathtt{T}}$.

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

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

$$\texttt{continuations}_{\underline{2}}(X) := [[X, \underline{2}], \underline{2}] = P(P(X))$$

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."

A look towards the future

In "The Category Theoretic Understanding of Universal Algebra: Lawvere Theories and Monads", Martin Hyland and John Power suggest that "computational effects might be seen as an instance and development of universal algebra." From this viewpoint:

- "Continuations would not be regarded as a computational effect but rather as a distinct notion. It would still have its own body of theory, and one would still study the relationship between it and computational effects; but perhaps it would not be regarded as a computational effect?"
- "Monads appear quite directly in the study of continuations. So perhaps the notion of monad might be seen as a generalised semantics of continuations?"

Review

Let $\mathtt{T}$ denote a notion of computation.

- 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

Thank you!