Understanding the classical monad-theory correspondence

Nathan Corbyn

University of Oxford

PUDDLE 22.05.2024

Objective

- Prove the equivalence:

$$\mathrm{Law} \simeq \mathrm{Mnd}_{\mathit{fin}}(\mathrm{Set})$$

Objective

- ~~Prove~~*Understand* the equivalence:

$$\mathrm{Law} \simeq \mathrm{Mnd}_{\mathit{fin}}(\mathrm{Set})$$

Plan

- Take three steps:

$$\begin{aligned}\mathrm{Law} &\simeq \mathrm{ProMnd}_{\times}(\mathbb{F}^{\mathrm{op}}) \\ &\simeq \mathrm{RMnd}(\mathbb{F} \hookrightarrow \mathrm{Set}) \\ &\simeq \mathrm{Mnd}_{\mathit{fin}}(\mathrm{Set})\end{aligned}$$

- Although not necessary to write down a proof, hopefully these all feel very natural by the end!
- Arkor's thesis takes this approach in extreme generality

Outline

Part I

- Lawvere theories
- Cartesian promonads
- $\mathrm{Law} \simeq \mathrm{ProMnd}_{\times}(\mathbb{F}^{\mathrm{op}})$

Part II

- Finitary monads
- Relative monads
- $\mathrm{RMnd}(\mathbb{F} \hookrightarrow \mathrm{Set}) \simeq \mathrm{Mnd}_{\mathit{fin}}(\mathrm{Set})$

Part III

- $\mathrm{ProMnd}_{\times}(\mathbb{F}^{\mathrm{op}}) \simeq \mathrm{RMnd}(\mathbb{F} \hookrightarrow \mathrm{Set})$

Part I

Lawvere theories

- Presentation invariant descriptions of algebraic theories
- Can be quite tricky to wrap your head around

Lawvere theories

- Consider a presentation of the theory of monoids:

$$u : 0 \qquad\qquad \oplus : 2$$

$$u \oplus \mathtt{x} = \mathtt{x}$$
$$\mathtt{x} \oplus u = \mathtt{x}$$
$$(\mathtt{x} \oplus \mathtt{y}) \oplus \mathtt{z} = \mathtt{x} \oplus (\mathtt{y} \oplus \mathtt{z})$$

Lawvere theories

- We get lots of *derivable* operations—e.g.,

$$\mathtt{x} \oplus (\mathtt{y} \oplus (\mathtt{z} \oplus \mathtt{w}))$$
$$\mathtt{x} \oplus \mathtt{x}$$
$$(u \oplus \mathtt{x}) \oplus (\mathtt{x} \oplus u)$$
$$(\mathtt{x} \oplus \mathtt{y}) \oplus (\mathtt{z} \oplus \mathtt{w})$$

- Some of these are *provably equal*:

$$\begin{aligned} \mathtt{x} \oplus \mathtt{x} &= (u \oplus \mathtt{x}) \oplus (\mathtt{x} \oplus u) \\ (\mathtt{x} \oplus \mathtt{y}) \oplus (\mathtt{z} \oplus \mathtt{w}) &= \mathtt{x} \oplus (\mathtt{y} \oplus (\mathtt{z} \oplus \mathtt{w})) \end{aligned}$$

- Important: Copying and discarding of variables is allowed

Lawvere theories

- **Lawvere's idea**: no matter how you present the theory, the same operations should be derivable and satisfy the same equations
- A Lawvere theory bundles derivable operations and their equations into a category

Lawvere theories

- What does this look like?
- For each $n \in \mathbb{N}$, write down the set of derivable operations in at most n variables, modulo provable equality:

$$T(n, 1)$$

- Extend this to all $m \in \mathbb{N}$ taking

$$\begin{aligned} T(n, 0) &= \{\star\} \\ T(n, m+1) &= T(n, m) \times T(n, 1) \end{aligned}$$

- In other words, $T(n, m)$ consists of tuples of m operations each in (at most) n variables

Lawvere theories

- **Idea**: $T(-, =)$ describes the hom-sets of a category.
- Composition is substitution!

$$\underbrace{T(n,m)}_{m \text{ operations in } n \text{ variables}} \times \underbrace{T(m,1)}_{1 \text{ operation in } m \text{ variables}}$$

$$\downarrow$$

$$\underbrace{T(n,1)}_{\text{substitute the } m \text{ variables for the } m \text{ derived operations}}$$

Lawvere theories

- We get more than just a category, we get a *cartesian* category:

$$n \xleftarrow{\langle \mathtt{x}_i \rangle_{i \in \underline{n}}} n + m \xrightarrow{\langle \mathtt{x}_{i+n} \rangle_{i \in \underline{m}}} m \qquad\qquad n \xrightarrow{\langle\rangle} 0$$

- This is intimately connected with the fact that we've allowed variables to be copied and discarded

Relationship to $\mathbb{F}^{\mathrm{op}}$

- Call $\mathbb{F}$ the category with objects $n \in \mathbb{N}$ and morphisms $\mathbb{F}(n, m) = \underline{n} \to \underline{m}$
- **Claim**: $\mathbb{F}^{\mathrm{op}}$ corresponds to the Lawvere theory determined by the empty presentation:
  - The only derivable operations are the variables:

$$\begin{aligned} T_\varnothing(n, 1) &= \underline{n} \\ &\cong \mathbb{F}(1, n) \end{aligned}$$

  - Extending,

$$\begin{aligned} T_\varnothing(n, 0) &= \{\star\} \cong \mathbb{F}(0, n) \\ T_\varnothing(n, m+1) &= T_\varnothing(n, m) \times T_\varnothing(n, 1) \\ &\cong \mathbb{F}(m, n) \times F(1, n) \\ &\cong \mathbb{F}(m+1, n) \end{aligned}$$

Relationship to $\mathbb{F}^{\mathrm{op}}$

- If we have two presentations $\Theta \subseteq \Theta'$, we get a (unique) corresponding identity-on-objects functor $T_\Theta \to T_{\Theta'}$
  - Hand-waving: All the operations derivable in the old theory are still derivable in the new theory, but might be identified via the new equations
- In particular, we always have a (unique) identity-on-objects functor

$$\mathbb{F}^{op} \to T_\Theta$$

- Moreover, this functor will always preserve products *strictly*
- We can use this to define Lawvere theories semantically!

Semantic definition

- **Definition**: a *Lawvere theory* is a category T equipped with a strictly product-preserving identity-on-objects functor $J : \mathbb{F}^{\mathrm{op}} \to T$
- We obtain a category Law of Lawvere theories and triangles:

$$\begin{tikzcd} T \arrow[rr, "F"] & & T' \\ & \mathbb{F}^{\mathrm{op}} \arrow[ul, "J"] \arrow[ur, "J'"'] & \end{tikzcd}$$

Promonads

- **Definition**: A promonad is a monoid in the category of endoprofunctors on a category $\mathcal{C}$

Promonads

- ~~**Definition**: A promonad is a monoid in the category of endoprofunctors on a category~~ $\cancel{\mathcal{C}}$

Promonads

- **Definition**: A promonad is a profunctor $P : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathrm{Set}$ equipped with maps

$$\mu : \int^{c:\mathcal{C}} P(-, c) \times P(c, =) \to P(-, =)$$
$$\eta : \mathcal{C}(-, =) \to P(-, =)$$

subject to...

Promonads

- ~~**Definition**: A promonad is a profunctor $P : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathrm{Set}$ equipped with maps~~

$$\mu : \int^{c:\mathcal{C}} P(-, c) \times P(c, =) \to P(-, =)$$

$$\eta : \mathcal{C}(-, =) \to P(-, =)$$

~~subject to...~~

Promonads

- Promonads are an extremely useful way to build new categories from old ones
- Promonads show up everywhere but aren't given the credit they deserve

Promonads

- Say we have some category $\mathcal{C}$
- We love the objects of $\mathcal{C}$
- But the morphisms are a bit of a disappointment:
  - Missing some maps
  - Not enough equations
  - Still important though!
- Promonads are a technical tool for describing the morphisms we wish we had and how they relate to the morphisms we've got right now
- If we set things up properly, we get a new category $\mathcal{D}$ with the same objects as $\mathcal{C}$ and an identity-on-objects functor $\mathcal{C} \to \mathcal{D}$

Promonads

- How do we do this?
- Start with a *profunctor* $P(-,=) : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathrm{Set}$
  - For each $x, y \in \mathcal{C}$, $P(x, y)$ is the hom-set you wish you had
  - The functorial actions tell you how to compose your dream maps with your disappointments
- Ask for a natural transformation

$$\eta : \mathcal{C}(-,=) \to P(-,=)$$

  - Every disappointment has something to live up to
  - Not necessarily injective

Promonads

- We've also got to explain how to compose our ideal maps
- For each $c \in \mathcal{C}$ we *want* to say we have a natural transformation:

$$\mu_c : P(-, c) \times P(c, =) \to P(-, =)$$

- But it's a bit more subtle!

Promonads

- We need all our composition operations to line up with each other:

$$
\begin{tikzcd}
x \arrow[r] \arrow[rr, bend right] \arrow[rrr, bend left=40] \arrow[rrr, bend right=40] & c \arrow[r, dotted] \arrow[rr, bend left] & c' \arrow[r] & y
\end{tikzcd}
$$

- The fancy way to say this is that we have a single natural transformation

$$\mu : \int^{c:\mathcal{C}} P(-, c) \times P(c, =) \to P(-, =)$$

- Jumping straight to coends is *missing the point*

Promonads

- Finally, we need a couple of laws to hold:
  - Composition should be associative
  - Lifting and composing should agree with acting

$$\begin{tikzcd} P(-,c) \times \mathcal{C}(c,=) \arrow[r, "P(-,c)\times\eta_c"] \arrow[dr, "\triangleleft_c"'] & P(-,c) \times P(c,=) \arrow[d, "\mu_c"] \\ & P(-,=) \end{tikzcd}$$

- We get a category $\mathrm{ProMnd}(\mathcal{C})$ whose objects are promonads on $\mathcal{C}$ and morphisms are natural transformations which respect the composition and lifting

Promonads

- What's this bought us?
- We certainly get a functor:

$$\mathrm{ProMnd}(\mathcal{C}) \to (\mathcal{C}/\mathrm{Cat})_{\mathsf{ioo}}$$

- In fact, we get an equivalence:

$$\mathrm{ProMnd}(\mathcal{C}) \simeq (\mathcal{C}/\mathrm{Cat})_{\mathsf{ioo}}$$

Cartesian promonads

- You may have spotted, we've got the following:

$$\mathrm{Law} = (\mathbb{F}^{\mathrm{op}}/\mathrm{Cat})_{\mathsf{ioo},\times} \hookrightarrow (\mathbb{F}^{\mathrm{op}}/\mathrm{Cat})_{\mathsf{ioo}} \simeq \mathrm{ProMnd}(\mathbb{F}^{\mathrm{op}})$$

- Can we restrict the right-hand side to get an equivalence?

Cartesian promonads

- We only want to consider promonads which induce cartesian functors
- Consider $(P : \mathbb{F} \times \mathbb{F}^{\mathrm{op}} \to \mathrm{Set}, \mu, \eta)$ a promonad on $\mathbb{F}^{\mathrm{op}}$
- Because the induced functor is identity-on-objects, it will strictly preserve products iff our dream maps still validate the universal properties
- In other words,

$$
\begin{aligned}
P(-,0) &\cong \top \\
P(-,n+m) &\cong P(-,n) \times P(-,m)
\end{aligned}
$$

naturally in m and n

Cartesian promonads

$$P(-,0) \cong \top$$
$$P(-,n+m) \cong P(-,n) \times P(-,m)$$

- Key point: this is the same as asking that the curried functor

$$P : \mathbb{F} \to [\mathbb{F}^{\mathrm{op}}, \mathrm{Set}]$$

  lands in cartesian functors
- We want *cartesian* profunctors:

$$P : \mathbb{F} \to [\mathbb{F}^{\mathrm{op}}, \mathrm{Set}]_\times$$

The first equivalence

$$\mathrm{Law} \simeq \mathrm{ProMnd}_{\times}(\mathbb{F}^{\mathrm{op}})$$

Part II

Monads

- A monad is a monoid in the category of endofunctors on a category $\mathcal{C}$

Monads

- ~~A monad is a monoid in the category of endofunctors on a category~~ $\cancel{\mathcal{C}}$

Monads

- A monad is a lax 2-functor $\mathbf{1} \to \mathrm{Cat}$

Monads

- ~~A monad is a lax 2-functor $\mathbf{1} \to \mathbb{C}\mathrm{at}$~~

Monads

- Monads are a technical tool for describing algebraic structures internal to general categories
- Rather than take a syntactic approach, monads are fundamentally semantically motivated
- The monad-theory correspondence for Set essentially says that the semantic approach and the syntactic approach are secretly the same
- *Mumble mumble technicalities...*

Monads

- Fundamental observation: an algebra is an object equipped with some operations we can somehow evaluate
- Take an object $x \in \mathcal{C}$, what does it mean to evaluate in x?
- Choose another object $Tx \in \mathcal{C}$ of 'computations' and a map

$$a : Tx \to x$$

Monads

- How do we make sure we've chosen a sensible notion of computation?
- First, we make $T : \mathcal{C} \to \mathcal{C}$ an endofunctor:
  - The notion of computation should be independent of the specific x I've chosen
  - Functoriality says that T can't 'see' x

Monads

- If we've already got a (generalised) element of x, we should have a 'do nothing' computation:

$$\eta_x : x \to Tx$$

- Similarly, if I have a computation that computes a computation, this should reduce to a single computation that works out what it needs to do and does it:

$$\mu_x : TTx \to Tx$$

- These should be natural (again, we shouldn't look at x):

$$\eta : 1_{\mathcal{C}} \to T \qquad\qquad \mu : TT \to T$$

Monads

- The monad laws express three more sensible properties of computation when you think in these terms!

$$
\begin{tikzcd}
Tx \arrow[r, "\eta_{Tx}"] \arrow[dr, "1_{Tx}"'] & TTx \arrow[d, dashed, "\mu_x"] & Tx \arrow[l, "T(\eta_x)"'] \arrow[dl, "1_{Tx}"] \\
& Tx &
\end{tikzcd}
\qquad
\begin{tikzcd}
TTTx \arrow[r, "T(\mu_x)"] \arrow[d, "\mu_{Tx}"'] & TTx \arrow[d, "\mu_x"] \\
TTx \arrow[r, "\mu_x"'] & Tx
\end{tikzcd}
$$

- We get a category $\mathrm{Mnd}(\mathcal{C})$ whose objects are monads on $\mathcal{C}$ and whose morphisms are natural transformations preserving all the structure

Finitary monads

- Disclaimer: this part is quite technical, so I'm going to brush over a lot of details, but hopefully the picture still comes out!

Finitary monads

- We're not always interested in notions of computation in the most general sense
- Sometimes we want to ensure that our computations are somehow 'finitely describable':
    - If we have a monad T on Set, we might hope that a computation $c \in TX$ can be described using at most finitely many elements of X
    - For example, computations might be formal sums of at most finitely many elements
- If we're looking for a connection with universal algebra, this is certainly a sensible restriction!

Finitary monads

- Category theory gives us some very technical, but very useful, notions of finiteness in general categories
- As we're only really interested in monads on Set today:
    - A monad T on Set is finitary iff for every set X and element $c \in TX$, there is a finite subset $X_0 \subseteq X$ through which c factors:

$$\begin{tikzcd} \{\star\} \arrow[rr, "c"] \arrow[dr, "c_0"'] & & TX \\ & TX_0 \arrow[ur] & \end{tikzcd}$$

- We'll call the full subcategory of $\mathrm{Mnd}(\mathrm{Set})$ spanned by finitary monads $\mathrm{Mnd}_{\mathit{fin}}(\mathrm{Set})$
- It only really matters what T does to finite sets!

Relative monads

- A monad relative to a functor $J : \mathcal{C} \to \mathcal{E}$ is a monoid in the skew-monoidal category $([\mathcal{C}, \mathcal{E}], \circ^J, J)$

Relative monads

- ~~A monad relative to a functor $J : \mathcal{C} \to \mathcal{E}$ is a monoid in the skew-monoidal category $([\mathcal{C}, \mathcal{E}], \circ^J, J)$~~

Relative monads

- Relative monads are a technical tool for describing notions of computation constrained to some particular diagram in a category
- The computations might form objects in a much larger category, but are only described for a (potentially) smaller system of objects

Relative monads

- What does this look like?
- First, pick your system of objects $J : \mathcal{C} \to \mathcal{E}$
- For each object $x \in \mathcal{C}$, define the computations $Tx \in \mathcal{E}$
- Again, we want 'do nothing' computations

$$\eta_x : Jx \to Tx$$

Relative monads

- This time, we can't necessarily build computations that compute computations
- However, we can introduce a mechanism for *sequencing* computations

$$(-)^{\dagger} : \mathcal{E}(Jx, Ty) \to \mathcal{E}(Tx, Ty)$$

  - I find it helps to think of maps $Jx \to Ty$ as computations with a 'parameter'

Relative monads

- Similar to monads, we have some sensible properties we expect to hold

$$(\eta_x)^\dagger = 1_x$$

$$\begin{tikzcd} & Tx \arrow[dr, "f^\dagger"] & \\ Jx \arrow[ur, "\eta_x"] \arrow[rr, "f"'] & & Ty \end{tikzcd}$$

Relative monads

$$Jx \xrightarrow{f} Ty \xrightarrow{g^\dagger} Tz$$

$$\begin{tikzcd}
Tx \arrow[rr, "(g^\dagger f)^\dagger"] \arrow[dr, "f^\dagger"'] & & Tz \\
& Ty \arrow[ur, "g^\dagger"'] &
\end{tikzcd}$$

- These laws automatically guarantee functoriality of T and naturality of η and $(-)^\dagger$!
- For each $J : \mathcal{C} \to \mathcal{E}$, we get a category $\mathrm{RMnd}(J)$ of monads relative to J and natural transformations preserving the structure

The second equivalence

$$\mathrm{RMnd}(\mathbb{F} \hookrightarrow \mathrm{Set}) \simeq \mathrm{Mnd}_{\mathit{fin}}(\mathrm{Set})$$

The picture

$$
\begin{tikzcd}
 & \mathrm{Set} \arrow[dr, "\mathrm{Lan}_J T"] & \\
\mathbb{F} \arrow[ur, hook, "J"] \arrow[rr, "T"'] \arrow[rr, phantom, shift left=4ex, "\Uparrow"] & & \mathrm{Set}
\end{tikzcd}
$$

Part III

The third equivalence

$$\mathrm{ProMnd}_{\times}(\mathbb{F}^{\mathrm{op}}) \simeq \mathrm{RMnd}(\mathbb{F} \hookrightarrow \mathrm{Set})$$

What we've got

- $P : \mathbb{F} \to [\mathbb{F}^{\mathrm{op}}, \mathrm{Set}]_\times$
- $\eta : \mathbb{F}^{\mathrm{op}}(-, =) \to P$
- $\mu : \int^{c:\mathcal{C}} P(-, c) \times P(c, =) \to P$
- ...

- $T : \mathbb{F} \to \mathrm{Set}$
- $\eta : J \to T$
- $(-)^\dagger : \mathrm{Set}(J(-), T(=)) \to \mathrm{Set}(T(-), T(=))$
- ...

Key ingredient

$$[\mathbb{F}^{\mathrm{op}}, \mathrm{Set}]_\times \simeq \mathrm{Set}$$

What we've got

- $P : \mathbb{F} \to [\mathbb{F}^{\mathrm{op}}, \mathrm{Set}]_\times$
- $\eta : \mathbb{F}^{\mathrm{op}}(-, =) \to P$
- $\mu : \int^{c:\mathcal{C}} P(-, c) \times P(c, =) \to P$
- ...

- $T : \mathbb{F} \to \mathrm{Set} \simeq [\mathbb{F}, \mathrm{Set}]_\times$
- $\cancel{\eta : J \to T}$ $\eta : \mathbb{F}^{\mathrm{op}}(-, =) \to T$
- $\cancel{(-)^\dagger : \mathrm{Set}(J(-), T(=)) \to \mathrm{Set}(T(-), T(=))}$

$$T(m, n) \cong [\mathbb{F}^{\mathrm{op}}, Set]_\times(\mathbb{F}^{\mathrm{op}}(n, -), Tm) \to [\mathbb{F}^{\mathrm{op}}, Set]_\times(Tn, Tm)$$

Summary

- We've covered a lot of ground:
  - Lawvere theories
  - Finitary monads
  - Relative monads
  - Promonads
  - Ways these all link up!
- Hopefully you've got some intuitions for some of these and can go away and look at the details

References

Thorsten Altenkirch, James Chapman, and Tarmo Uustalu.
Monads need not be endofunctors.
Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6014 LNCS:297–311, 2010.

Nathanael Arkor.
Monadic and higher-order structure.
2022.

F. William Lawvere.
Functorial semantics of algebraic theories and some algebraic problems in the context of functorial semantics of algebraic theories, 1963.