

Chapter 7: Computations lifted to a functor context II. Monads

Part 2: Laws and structure of semimonads

Sergei Winitzki

Academy by the Bay

2018-03-11

Semimonad laws I: The intuitions

What properties of functor block programs do we expect to have?

- In $x \leftarrow c$, the value of x will *go over items* held in container c
- Manipulating items in container is followed by a generator:

$x \leftarrow \text{cont1}$	$y \leftarrow \text{cont1}$
$y = f(x)$	$\quad \text{.map}(x \Rightarrow f(x))$
$z \leftarrow \text{cont2}(y)$	$z \leftarrow \text{cont2}(y)$

$\text{cont1.flatMap}(x \Rightarrow \text{cont2}(f(x))) = \text{cont1.map}(f).\text{flatMap}(y \Rightarrow \text{cont2}(y))$

- Manipulating items in container is preceded by a generator:

$x \leftarrow \text{cont1}$	$x \leftarrow \text{cont1}$
$y \leftarrow \text{cont2}(x)$	$z \leftarrow \text{cont2}(x)$
$z = f(y)$	$\quad \text{.map}(f)$

$\text{cont1.flatMap}(\text{cont2}).\text{map}(f) = \text{cont1.flatMap}(x \Rightarrow \text{cont2}(x).\text{map}(f))$

- After $x \leftarrow \text{cont}$, further computations will use *all those* x

$x \leftarrow \text{cont}$	$y \leftarrow \text{for } \{ x \leftarrow \text{cont}$
$y \leftarrow p(x)$	$\quad yy \leftarrow p(x) \} \text{ yield } yy$
$z \leftarrow \text{cont2}(y)$	$z \leftarrow \text{cont2}(y)$

$\text{cont.flatMap}(x \Rightarrow p(x).\text{flatMap}(\text{cont2})) = \text{cont.flatMap}(p).\text{flatMap}(\text{cont2})$

Semimonad laws II: The laws for `flatMap`

To get a more concise notation, use `flm` instead of `flatMap`

A **semimonad** S^A has $\text{flm}^{[S, A, B]} : (A \Rightarrow S^B) \Rightarrow S^A \Rightarrow S^B$ with 3 laws:

❶ $\text{flm} (f^{A \Rightarrow B} \circ g^{B \Rightarrow S^C}) = \text{fmap } f \circ \text{flm } g$ (naturality in A)

$$\begin{array}{ccc} & S^B & \\ \text{fmap } f^{A \Rightarrow B} \nearrow & & \searrow \text{flm } g^{B \Rightarrow S^C} \\ S^A & \xRightarrow{\text{flm } (f^{A \Rightarrow B} \circ g^{B \Rightarrow S^C})} & S^C \end{array}$$

❷ $\text{flm} (f^{A \Rightarrow S^B} \circ \text{fmap } g^{B \Rightarrow C}) = \text{flm } f \circ \text{fmap } g$ (naturality in B)

$$\begin{array}{ccc} & S^B & \\ \text{flm } f^{A \Rightarrow S^B} \nearrow & & \searrow \text{fmap } g^{B \Rightarrow C} \\ S^A & \xRightarrow{\text{flm } (f^{A \Rightarrow S^B} \circ \text{fmap } g^{B \Rightarrow C})} & S^C \end{array}$$

❸ $\text{flm} (f^{A \Rightarrow S^B} \circ \text{flm } g^{B \Rightarrow S^C}) = \text{flm } f \circ \text{flm } g$ (composition)

$$\begin{array}{ccc} & S^B & \\ \text{flm } f^{A \Rightarrow S^B} \nearrow & & \searrow \text{flm } g^{B \Rightarrow S^C} \\ S^A & \xRightarrow{\text{flm } (f^{A \Rightarrow S^B} \circ \text{flm } g^{B \Rightarrow S^C})} & S^C \end{array}$$

Is there a shorter formulation of the laws?

Semimonad laws III: The laws for `flatten`

The methods `flatten` (denoted by `ftn`) and `flatMap` are equivalent:

$$\text{ftn}^{[S,A]} : S^{S^A} \Rightarrow S^A = \text{flm}^{[S,S^A,A]}(m^{S^A} \Rightarrow m)$$

$$\text{flm}(f^{A \Rightarrow S^B}) = \text{fmap } f \circ \text{ftn}$$

A commutative triangle diagram. The top-left node is S^A , the top-right node is S^{S^B} , and the bottom-right node is S^B . An arrow labeled $\text{fmap } f^{A \Rightarrow S^B}$ points from S^A to S^{S^B} . An arrow labeled ftn points from S^{S^B} to S^B . A bottom arrow labeled $\text{flm}(f^{A \Rightarrow S^B})$ points from S^A to S^B .

It turns out that `flatten` has only 2 laws:

- ❶ $\text{fmap}(\text{fmap } f^{A \Rightarrow B}) \circ \text{ftn}^{[S,B]} = \text{ftn}^{[S,A]} \circ \text{fmap } f$ (naturality)

A commutative diagram with four nodes. The top-left node is S^{S^A} , the top-right node is S^{S^B} , the bottom-left node is S^A , and the bottom-right node is S^B . An arrow labeled $\text{fmap}(\text{fmap } f^{A \Rightarrow B})$ points from S^{S^A} to S^{S^B} . An arrow labeled $\text{ftn}^{[S,B]}$ points from S^{S^B} to S^B . An arrow labeled $\text{ftn}^{[S,A]}$ points from S^{S^A} to S^A . An arrow labeled $\text{fmap } f^{A \Rightarrow B}$ points from S^A to S^B .

- ❷ $\text{fmap}(\text{ftn}^{[S]}) \circ \text{ftn}^{[S^S]} = \text{ftn}^{[S^S]} \circ \text{ftn}^{[S]}$ (associativity)

A commutative diagram with four nodes. The top-left node is $S^{S^{S^A}}$, the top-right node is S^{S^A} , the bottom-left node is S^{S^A} , and the bottom-right node is S^A . An arrow labeled $\text{fmap}(\text{ftn}^{[S]})$ points from $S^{S^{S^A}}$ to S^{S^A} . An arrow labeled $\text{ftn}^{[S]}$ points from S^{S^A} to S^A . An arrow labeled $\text{ftn}^{[S^S]}$ points from $S^{S^{S^A}}$ to S^{S^A} . An arrow labeled $\text{ftn}^{[S]}$ points from S^{S^A} to S^A .

Equivalence of a natural transformation and a “lifting”

- Equivalence of flm and ftn : $\text{ftn} = \text{flm}(\text{id})$; $\text{flm } f = \text{fmap } f \circ \text{ftn}$
- We saw this before: $\text{deflate} = \text{fmapOpt}(\text{id})$; $\text{fmapOpt } f = \text{fmap } f \circ \text{deflate}$
 - ▶ Is there a general pattern where two such functions are equivalent?
- Let $\text{tr} : F^{G^A} \Rightarrow F^A$ be a natural transformation (F and G are functors)
- Define $\text{ftr} : (A \Rightarrow G^B) \Rightarrow F^A \Rightarrow F^B$ by $\text{ftr } f = \text{fmap } f \circ \text{tr}$
- It follows that $\text{tr} = \text{ftr}(\text{id})$, and we have equivalence between tr and ftr :

$$\text{tr} : F^{G^A} \Rightarrow F^A = \text{ftr}(m^{G^A} \Rightarrow m)$$

$$\text{ftr}(f^{A \Rightarrow G^B}) = \text{fmap } f \circ \text{tr}$$

$$\begin{array}{ccc} & F^{G^B} & \\ \text{fmap } f^{A \Rightarrow G^B} \nearrow & & \searrow \text{tr} \\ F^A & \xRightarrow{\text{ftr}(f^{A \Rightarrow G^B})} & F^B \end{array}$$

- An automatic law for ftr (“naturality in A ”) follows from the definition:
 $\text{fmap } g \circ \text{ftr } f = \text{fmap } g \circ \text{fmap } f \circ \text{tr} = \text{fmap}(g \circ f) \circ \text{tr} = \text{ftr}(g \circ f)$
 - ▶ This is why tr has *one law fewer* than ftr
- To demonstrate equivalence in the direction $\text{ftr} \rightarrow \text{tr}$: start with an arbitrary ftr satisfying “naturality in A ”, then obtain $\text{tr} = \text{ftr}(\text{id})$ from it, then verify $\text{ftr } f = \text{fmap } f \circ \text{tr}$ with that tr : $\text{fmap } f \circ \text{ftr}(\text{id}) = \text{ftr}(f \circ \text{id}) = \text{ftr } f$

Semimonad laws IV: Deriving the laws for `flatten`

Denote for brevity $q^\uparrow \equiv \text{fmap}^{[S]} q$ for any function q

Express $\text{flm } f = f^\uparrow \circ \text{ftn}$ and substitute that into flm 's 3 laws:

- ❶ $\text{flm } (f \circ g) = f^\uparrow \circ \text{flm } g$ gives $(f \circ g)^\uparrow \circ \text{ftn} = f^\uparrow \circ g^\uparrow \circ \text{ftn}$
– this law holds automatically due to functor composition law
 - ❷ $\text{flm } (f \circ g^\uparrow) = \text{flm } f \circ g^\uparrow$ gives $(f \circ h)^\uparrow \circ \text{ftn} = f^\uparrow \circ \text{ftn} \circ h$;
using the functor composition law, we reduce this to $h^\uparrow \circ \text{ftn} = \text{ftn} \circ h$ – this is the naturality law
 - ❸ $\text{flm } (f \circ \text{flm } g) = \text{flm } f \circ \text{flm } g$ with functor composition law gives $f^\uparrow \circ g^{\uparrow\uparrow} \circ \text{ftn}^\uparrow \circ \text{ftn} = f^\uparrow \circ \text{ftn} \circ g^\uparrow \circ \text{ftn}$; using ftn 's naturality and omitting the common factor $f^\uparrow \circ g^{\uparrow\uparrow}$, we get $\text{ftn}^\uparrow \circ \text{ftn} = \text{ftn} \circ \text{ftn}$ – associativity law
- `flatten` has the simplest type signature *and* the fewest laws
 - It is usually easy to check naturality!
 - ▶ **Parametricity theorem:** Any *fully parametric* code for a function of type $F^A \Rightarrow G^A$ implements a natural transformation
 - Checking `flatten`'s associativity needs a lot more work

The `cats` library has a `FlatMap` type class, defining `flatten` via `flatMap`

Checking the associativity law for standard monads

- Implement `flatten` for these functors and check the laws (see code):
 - ▶ `Option` monad: $F^A \equiv 1 + A$; $\text{ftn} : 1 + (1 + A) \Rightarrow 1 + A$
 - ▶ `Either` monad: $F^A \equiv Z + A$; $\text{ftn} : Z + (Z + A) \Rightarrow Z + A$
 - ▶ `List` monad: $F^A \equiv \text{List}^A$; $\text{ftn} : \text{List}^{\text{List}^A} \Rightarrow \text{List}^A$
 - ▶ `Writer` monad: $F^A \equiv A \times W$; $\text{ftn} : (A \times W) \times W \Rightarrow A \times W$
 - ▶ `Reader` monad: $F^A \equiv R \Rightarrow A$; $\text{ftn} : (R \Rightarrow (R \Rightarrow A)) \Rightarrow R \Rightarrow A$
 - ▶ `State`: $F^A \equiv S \Rightarrow A \times S$; $\text{ftn} : (S \Rightarrow (S \Rightarrow A \times S)) \times S \Rightarrow S \Rightarrow A \times S$
 - ▶ `Continuation` monad: $F^A \equiv (A \Rightarrow R) \Rightarrow R$;
 $\text{ftn} : (((A \Rightarrow R) \Rightarrow R) \Rightarrow R) \Rightarrow (A \Rightarrow R) \Rightarrow R$
- Code implementing these `flatten` functions is *fully parametric* in A
 - ▶ Naturality of these functions follows from parametricity theorem
 - ▶ Associativity needs to be checked for each monad!
- Example of a useful semimonad that is *not* a full monad:
 - ▶ $F^A \equiv A \times V \times W$; $\text{ftn}((a \times v_1 \times w_1) \times v_2 \times w_2) = a \times v_1 \times w_2$
- Examples of *non-associative* (i.e. wrong) implementations of `flatten`:
 - ▶ $F^A \equiv A \times W \times W$; $\text{ftn}((a \times v_1 \times v_2) \times w_1 \times w_2) = a \times w_2 \times w_1$
 - ▶ $F^A \equiv \text{List}^A$, but `flatten` concatenates the nested lists in reverse order

1 Implement

Structure of semimonads

How to recognize a semimonad by its type?

Intuition from `flatten`: reshuffle data in F^{F^A} to fit into F^A

Some constructions of exponential-polynomial semimonads:

- ① $F^A = Z$ (constant functor) for a fixed type Z
- ② $F^A \equiv Z + A \times W$ for a fixed type Z and a semigroup W
- ③ $F^A \equiv G^A \times H^A$ for any semimonads G^A and H^A
- ④ $F^A \equiv A + G^A$ for any semimonad G^A
- ⑤ $F^A \equiv A + G^{F^A}$ (recursive) for any functor G^A
- ⑥ $F^A \equiv G^A + G^{F^A}$ (recursive) for any functor G^A
- ⑦ $F^A \equiv H^A \Rightarrow A \times G^A$ for any contrafunctor H^A and semimonad G^A

* Worked examples II: Constructions of filterable functors I

(2) The `fmapOpt` laws hold for $F^A \times G^A$ if they hold for F^A and G^A

- For $f^{A \Rightarrow 1+B}$, get $\text{fmapOpt}_F(f) : F^A \Rightarrow F^B$ and $\text{fmapOpt}_G(f) : G^A \Rightarrow G^B$
- Define $\text{fmapOpt}_{F \times G} f \equiv p^{F^A} \times q^{G^A} \Rightarrow \text{fmapOpt}_F(f)(p) \times \text{fmapOpt}_G(f)(q)$
- Identity law: $f = \text{id}_{\diamond_{\text{Opt}}}$, so $\text{fmapOpt}_F f = \text{id}$ and $\text{fmapOpt}_G f = \text{id}$
 - ▶ Hence we get $\text{fmapOpt}_{F+G}(f)(p \times q) = \text{id}(p) \times \text{id}(q) = p \times q$
- Composition law:

$$\begin{aligned} & (\text{fmapOpt}_{F \times G} f_1 \circ \text{fmapOpt}_{F+G} f_2)(p \times q) \\ &= \text{fmapOpt}_{F \times G}(f_2) (\text{fmapOpt}_F(f_1)(p) \times \text{fmapOpt}_G(f_1)(q)) \\ &= (\text{fmapOpt}_F f_1 \circ \text{fmapOpt}_F f_2)(p) \times (\text{fmapOpt}_G f_1 \circ \text{fmapOpt}_G f_2)(q) \\ &= \text{fmapOpt}_F(f_1 \diamond_{\text{Opt}} f_2)(p) \times \text{fmapOpt}_G(f_1 \diamond f_2)(q) \\ &= \text{fmapOpt}_{F \times G}(f_1 \diamond_{\text{Opt}} f_2)(p \times q) \end{aligned}$$

- Exactly the same proof as that for functor property for $F^A \times G^A$
 - ▶ this is because `fmapOpt` corresponds to a generalized functor
- New proofs are necessary only when using non-filterable functors
 - ▶ these are used in constructions 4 – 6

* Worked examples II: Constructions of filterable functors II

(5) The `fmapOpt` laws hold for $F^A \equiv 1 + A \times G^A$ if they hold for G^A

- For $f^{A \Rightarrow 1+B}$, get $\text{fmapOpt}_G(f) : G^A \Rightarrow G^B$
- Define $\text{fmapOpt}_F(f)(1 + a^A \times q^{G^A})$ by returning $0 + b \times \text{fmapOpt}_G(f)(q)$ if the argument is $0 + a \times q$ and $f(a) = 0 + b$, and returning $1 + 0$ otherwise
- Identity law: $f = \text{id}_{\text{Opt}}$, so $f(a) = 0 + a$ and $\text{fmapOpt}_G f = \text{id}$
 - ▶ Hence we get $\text{fmapOpt}_F(\text{id}_{\text{Opt}})(1 + a \times q) = 1 + a \times q$
- Composition law: need only to check for arguments $0 + a \times q$, and only when $f_1(a) = 0 + b$ and $f_2(b) = 0 + c$, in which case $(f_1 \diamond_{\text{Opt}} f_2)(a) = 0 + c$; then

$$\begin{aligned} & (\text{fmapOpt}_F f_1 \circ \text{fmapOpt}_F f_2)(0 + a \times q) \\ &= \text{fmapOpt}_F(f_2) (\text{fmapOpt}_F(f_1)(0 + a \times q)) \\ &= \text{fmapOpt}_F(f_2) (0 + b \times \text{fmapOpt}_G(f_1)(q)) \\ &= 0 + c \times (\text{fmapOpt}_G f_1 \circ \text{fmapOpt}_G f_2)(q) \\ &= 0 + c \times \text{fmapOpt}_G(f_1 \diamond_{\text{Opt}} f_2)(q) \\ &= \text{fmapOpt}_F(f_1 \diamond_{\text{Opt}} f_2)(0 + a \times q) \end{aligned}$$

This is a “greedy filter”: if $f(a)$ is empty, will delete all data in G^A

* Worked examples II: Constructions of filterable functors III

(6) The `fmapOpt` laws hold for $F^A \equiv G^A + A \times F^A$ if they hold for G^A

- For $f^{A \Rightarrow 1+B}$, we have $\text{fmapOpt}_G(f) : G^A \Rightarrow G^B$ and $\text{fmapOpt}'_F(f) : F^A \Rightarrow F^B$ (for use in recursive arguments as the inductive assumption)
- Define $\text{fmapOpt}_F(f)(q^{G^A} + a^A \times p^{F^A})$ by returning $0 + \text{fmapOpt}'_F(f)(p)$ if $f(a) = 1 + 0$, and $\text{fmapOpt}_G(f)(q) + b \times \text{fmapOpt}'_F(f)(p)$ otherwise
- Identity law: $\text{id}_{\diamond_{\text{Opt}}}(x) \neq 1 + 0$, so $\text{fmapOpt}_F(\text{id}_{\diamond_{\text{Opt}}})(q + a \times p) = q + a \times p$
- Composition law:
 $(\text{fmapOpt}_F(f_1) \circ \text{fmapOpt}_F(f_2))(q + a \times p) = \text{fmapOpt}_F(f_1 \diamond_{\text{Opt}} f_2)(q + a \times p)$
- For arguments $q + 0$, the laws for G^A hold; so assume arguments $0 + a \times p$. When $f_1(a) = 0 + b$ and $f_2(b) = 0 + c$, the proof of the previous example will go through. So we need to consider the two cases $f_1(a) = 1 + 0$ and $f_1(a) = 0 + b$, $f_2(b) = 1 + 0$
- If $f_1(a) = 1 + 0$ then $(f_1 \diamond_{\text{Opt}} f_2)(a) = 1 + 0$; to show $\text{fmapOpt}'_F(f_2)(\text{fmapOpt}'_F(f_1)(p)) = \text{fmapOpt}'_F(f_1 \diamond_{\text{Opt}} f_2)(p)$, use the inductive assumption about $\text{fmapOpt}'_F$ on p
- If $f_1(a) = 0 + b$ and $f_2(b) = 1 + 0$ then $(f_1 \diamond_{\text{Opt}} f_2)(a) = 1 + 0$; to show $\text{fmapOpt}_F(f_2)(0 + b \times \text{fmapOpt}'_F(f_1)(p)) = \text{fmapOpt}'_F(f_1 \diamond_{\text{Opt}} f_2)(p)$, rewrite $\text{fmapOpt}_F(f_2)(0 + b \times \text{fmapOpt}'_F(f_1)(p)) = \text{fmapOpt}'_F(f_2)(\text{fmapOpt}'_F(f_1)(p))$ and again use the inductive assumption about $\text{fmapOpt}'_F$ on p

This is a “list-like filter”: if $f(a)$ is empty, will recurse into nested F^A data

Worked examples II: Constructions of filterable functors IV

Use known filterable constructions to show that

$F^A \equiv (\text{Int} \times \text{String}) \Rightarrow (1 + \text{Int} \times A + A \times (1 + A) + (\text{Int} \Rightarrow 1 + A + A \times A \times \text{String}))$
is a filterable functor

- Instead of implementing `Filterable` and verifying laws by hand, we analyze the structure of this data type and use known constructions
- Define some auxiliary functors that are parts of the structure of F^A ,
 - ▶ $R_1^A = (\text{Int} \times \text{String}) \Rightarrow A$ and $R_2^A = \text{Int} \Rightarrow A$
 - ▶ $G^A = 1 + \text{Int} \times A + A \times (1 + A)$ and $H^A = 1 + A + A \times A \times \text{String}$
- Now we can rewrite $F^A = R_1 [G^A + R_2 [H^A]]$
 - ▶ G^A is filterable by construction 5 because it is of the form $G^A = 1 + A \times K^A$ with filterable functor $K^A = 1 + \text{Int} + A$
 - ▶ K^A is of the form $1 + A + X$ with constant type X , so it is filterable by constructions 1 and 3 with the `Option` functor $1 + A$
 - ▶ H^A is filterable by construction 5 with $H^A = 1 + A \times (1 + A \times \text{String})$, while $1 + A \times \text{String}$ is filterable by constructions 5 and 1
- Constructions 3 and 4 show that $R_1 [G^A + R_2 [H^A]]$ is filterable

Note that there are more than one way of implementing `Filterable` here

* Exercises II

- 1 Implement a `Filterable` instance for `type F[T] = G[H[T]]` assuming that the functor `H[T]` already has a `Filterable` instance (construction 4). Verify the laws rigorously (i.e. by calculations, not tests).
- 2 For `type F[T] = Option[Int \Rightarrow Option[(T, T)]]`, implement a `Filterable` instance. Show that the filterable laws hold by using known filterable constructions (avoiding explicit proofs or tests).
- 3 Implement a `Filterable` instance for $F^A \equiv G^A + \text{Int} \times A \times A \times F^A$ (recursive) for a filterable functor G^A . Verify the laws rigorously.
- 4 Show that $F^A = 1 + A \times G^A$ is in general *not* filterable if G^A is an arbitrary (non-filterable) functor; it is enough to give an example.
- 5 Show that $F^A = 1 + G^A + H^A$ is filterable if $1 + G^A$ and $1 + H^A$ are filterable (even when G^A and H^A are by themselves not filterable).
- 6 Show that the functor $F^A = A + (\text{Int} \Rightarrow A)$ is not filterable.
- 7 Show that one can define `deflate`: $C^{1+A} \Rightarrow C^A$ for any contrafunctor C^A (not necessarily filterable), similarly to how one can define `inflate`: $F^A \Rightarrow F^{1+A}$ for any functor F^A (not necessarily filterable).