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

Part 2: Laws and structure of monads and semimonads

Sergei Winitzki

Academy by the Bay

2018-05-27

Semimonad laws I: The intuitions

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

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

```
x ← cont1                         y ← cont1
y = f(x)                                 .map(x ⇒ f(x))
z ← cont2(y)                      z ← cont2(y)
```

```
cont1.flatMap(x ⇒ cont2(f(x))) = cont1.map(f).flatMap(y ⇒ cont2(y))
```

- Manipulating items in container is preceded by a generator:

```
x ← cont1                         x ← cont1
y ← cont2(x)                      z ← cont2(x)
z = f(y)                                 .map(f)
```

```
cont1.flatMap(cont2).map(f) = cont1.flatMap(x ⇒ cont2(x).map(f))
```

- Within a generator, `for {...} yield` can be inlined:

```
x ← cont                          yy ← for { x ← cont
y ← p(x)                                     y ← p(x) } yield y
z ← cont2(y)                      z ← cont2(yy)
```

```
cont.flatMap(x ⇒ p(x).flatMap(cont2)) = cont.flatMap(p).flatMap(cont2)
```

For brevity, write flm instead of `flatMap`

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

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

$$\begin{tikzcd} & S^B \arrow[dr, "\text{flm}\,g^{B\Rightarrow S^C}"] & \\ S^A \arrow[ur, "\text{fmap}\,f^{A\Rightarrow B}"] \arrow[rr, "\text{flm}\,(f^{A\Rightarrow B}\circ g^{B\Rightarrow S^C})"'] & & S^C \end{tikzcd}$$

2. $\text{flm}\left(f^{A\Rightarrow S^B} \circ \text{fmap}\,g^{B\Rightarrow C}\right) = \text{flm}\,f \circ \text{fmap}\,g$ (naturality in B)

$$\begin{tikzcd} & S^B \arrow[dr, "\text{fmap}\,g^{B\Rightarrow C}"] & \\ S^A \arrow[ur, "\text{flm}\,f^{A\Rightarrow S^B}"] \arrow[rr, "\text{flm}\,(f^{A\Rightarrow S^B}\circ \text{fmap}\,g^{B\Rightarrow C})"'] & & S^C \end{tikzcd}$$

3. $\text{flm}\left(f^{A\Rightarrow S^B} \circ \text{flm}\,g^{B\Rightarrow S^C}\right) = \text{flm}\,f \circ \text{flm}\,g$ (associativity)

$$\begin{tikzcd} & S^B \arrow[dr, "\text{flm}\,g^{B\Rightarrow S^C}"] & \\ S^A \arrow[ur, "\text{flm}\,f^{A\Rightarrow S^B}"] \arrow[rr, "\text{flm}\left(f^{A\Rightarrow S^B}\circ \text{flm}\,g^{B\Rightarrow S^C}\right)"'] & & S^C \end{tikzcd}$$

Is there a shorter and clearer formulation of these laws?

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

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

$$\mathrm{flm}\left(f^{A\Rightarrow S^B}\right) \equiv \mathrm{fmap}\, f \circ \mathrm{ftn}$$

$$\begin{tikzcd}
 & S^{S^B} \arrow[dr, "\mathrm{ftn}"] & \\
S^A \arrow[ur, "\mathrm{fmap}\, f^{A\Rightarrow S^B}"] \arrow[rr, "\mathrm{flm}\left(f^{A\Rightarrow S^B}\right)"'] & & S^B
\end{tikzcd}$$

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

1. $\mathrm{fmap}\big(\mathrm{fmap}\, f^{A\Rightarrow B}\big) \circ \mathrm{ftn}^{[B]} = \mathrm{ftn}^{[A]} \circ \mathrm{fmap}\, f$ (naturality)

$$\begin{tikzcd}
 & S^{S^B} \arrow[dr, "\mathrm{ftn}^{[B]}"] & \\
S^{S^A} \arrow[ur, "\mathrm{fmap}\left(\mathrm{fmap}\, f^{A\Rightarrow B}\right)"] \arrow[dr, "\mathrm{ftn}^{[A]}"'] & & S^B \\
 & S^A \arrow[ur, "\mathrm{fmap}\, f^{A\Rightarrow B}"'] &
\end{tikzcd}$$

2. $\mathrm{fmap}\,(\mathrm{ftn}^{[A]}) \circ \mathrm{ftn}^{[A]} = \mathrm{ftn}^{[S^A]} \circ \mathrm{ftn}^{[A]}$ (associativity)

$$\begin{tikzcd}
 & S^{S^A} \arrow[dr, "\mathrm{ftn}^{[A]}"] & \\
S^{S^{S^A}} \arrow[ur, "\mathrm{fmap}\,(\mathrm{ftn}^{[A]})"] \arrow[dr, "\mathrm{ftn}^{[S^A]}"'] & & S^A \\
 & S^{S^A} \arrow[ur, "\mathrm{ftn}^{[A]}"'] &
\end{tikzcd}$$

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:

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

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

$$\begin{tikzcd} & F^{G^B} \arrow[dr, "\text{tr}"] & \\ F^A \arrow[ur, "\text{fmap}\,f^{A\Rightarrow G^B}"] \arrow[rr, "\text{ftr}\left(f^{A\Rightarrow G^B}\right)"'] & & F^B \end{tikzcd}$$

- 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 always 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; $\text{fmap}\,f \circ \text{ftr}\,(\text{id}) = \text{ftr}\,(f \circ id) = \text{ftr}\,f$

Semimonad laws IV: Deriving the laws for `flatten`

Denote for brevity $q^{\uparrow} \equiv \text{fmap}\, q$ for any function q ("lifting" $q^{A \Rightarrow B}$ to S)
Express $\text{flm}\, f = f^{\uparrow} \circ \text{ftn}$ and substitute that into flm's 3 laws:

1. $\text{flm}\,(f \circ g) = f^{\uparrow} \circ \text{flm}\, g$ 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
2. $\text{flm}\left(f \circ g^{\uparrow}\right) = \text{flm}\, f \circ g^{\uparrow}$ gives $\left(f \circ g^{\uparrow}\right)^{\uparrow} \circ \text{ftn} = f^{\uparrow} \circ \text{ftn} \circ g^{\uparrow}$;
   using the functor composition law, we reduce this to
   $g^{\uparrow\uparrow} \circ \text{ftn} = \text{ftn} \circ g^{\uparrow}$ – this is the naturality law
3. $\text{flm}\,(f \circ \text{flm}\, g) = \text{flm}\, f \circ \text{flm}\, g$ 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 *pure, fully parametric* code for a function of type $F^A \Rightarrow G^A$ will implement 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: $F^A \equiv 1 + A$; ftn : $1 + (1 + A) \Rightarrow 1 + A$
  - `Either` monad: $F^A \equiv Z + A$; ftn : $Z + (Z + A) \Rightarrow Z + A$
  - `List` monad: $F^A \equiv \text{List}^A$; ftn : $\text{List}^{\text{List}^A} \Rightarrow \text{List}^A$
  - Writer monad: $F^A \equiv A \times W$; ftn : $(A \times W) \times W \Rightarrow A \times W$
  - Reader monad: $F^A \equiv R \Rightarrow A$; ftn : $(R \Rightarrow (R \Rightarrow A)) \Rightarrow R \Rightarrow A$
  - State: $F^A \equiv S \Rightarrow A \times S$; ftn : $(S \Rightarrow (S \Rightarrow A \times S) \times S) \Rightarrow S \Rightarrow A \times S$
  - Continuation monad: $F^A \equiv (A \Rightarrow R) \Rightarrow R$;
    ftn : $((((A \Rightarrow R) \Rightarrow R) \Rightarrow R) \Rightarrow R) \Rightarrow (A \Rightarrow R) \Rightarrow R$
- 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

Motivation for monads

- Monads represent values with a "special computational context"
- Specific monads will have methods to create various contexts
- Monadic composition will "combine" the contexts associatively
- It is generally useful to have an "empty context" available:

$$\text{pure} : A \Rightarrow M^A$$

Adding the empty context to another context should be a no-op

- Empty context is followed by a generator:

```
y ← pure(x)
z ← cont(y)
```

```
y = x
z ← cont(y)
```

`pure(x).flatMap(y ⇒ cont(y)) = cont(x)`

$\text{pure} \circ \text{flm}\, f = f$ – left identity

- Empty context is preceded by a generator:

```
x ← cont
y ← pure(x)
```

```
x ← cont
y = x
```

`cont.flatMap(x ⇒ pure(x)) = cont`

$\text{flm}\,(\text{pure}) = \text{id}$ – right identity

The monad laws formulated in terms of `pure` and `flatten`

- Naturality law for `pure`: $f \circ \text{pure} = \text{pure} \circ \text{fmap}\, f$

$$\begin{tikzcd} & B \arrow[dr, "\text{pure}^{[B]}"] & \\ A \arrow[ur, "f^{A\Rightarrow B}"] \arrow[dr, "\text{pure}^{[A]}"'] & & S^B \\ & S^A \arrow[ur, "\text{fmap}\, f^{A\Rightarrow B}"'] & \end{tikzcd}$$

- Left identity: $\text{pure} \circ \text{flm}\, f = \text{pure} \circ \text{fmap}\, f \circ \text{ftn} = f \circ \text{pure} \circ \text{ftn} = f$ requires that $\text{pure} \circ \text{ftn} = \text{id}$ (both sides applied to S^A)

$$\begin{tikzcd} & S^{S^A} \arrow[dr, "\text{ftn}^{[A]}"] & \\ S^A \arrow[ur, "\text{pure}^{[S^A]}"] \arrow[rr, "\text{id}"'] & & S^A \end{tikzcd}$$

- Right identity: $\text{flm}\,(\text{pure}) = \text{fmap}\,(\text{pure}) \circ \text{ftn} = \text{id}^{S^A \Rightarrow S^A}$

$$\begin{tikzcd} & S^{S^A} \arrow[dr, "\text{ftn}^{[A]}"] & \\ S^A \arrow[ur, "\text{fmap}(\text{pure}^{[A]})"] \arrow[rr, "\text{id}"'] & & S^A \end{tikzcd}$$

Formulating laws via Kleisli functions

- Recall: we formulated the laws of filterables via `fmapOpt`
  - type signature of fmapOpt : $(A \Rightarrow 1 + B) \Rightarrow S^A \Rightarrow S^B$
  - and then we had to compose functions of types $A \Rightarrow 1 + B$ via $\diamond_{\text{Opt}}$
- Here we have flm : $\left(A \Rightarrow S^B\right) \Rightarrow S^A \Rightarrow S^B$ instead of `fmapOpt`
- Can we compose **Kleisli functions** with "twisted" types $A \Rightarrow S^B$?
- Use flm to define **Kleisli composition**: $f^{A \Rightarrow S^B} \diamond g^{B \Rightarrow S^C} \equiv f \circ \text{flm}\, g$
- Define **Kleisli identity** $\text{id}_\diamond$ of type $A \Rightarrow S^A$ as $\text{id}_\diamond \equiv \text{pure}$
- Composition law: $\text{flm}\,(f \diamond g) = \text{flm}\, f \circ \text{flm}\, g$ (same as for `fmapOpt`)
  - Shows that `flatMap` is a "lifting" of $A \Rightarrow S^B$ to $S^A \Rightarrow S^B$
- These laws are similar to functor "lifting" laws...
  - except that $\diamond$ is used for composing Kleisli functions
- What are the properties of $\diamond$?
  - Exactly similar to the properties of function composition $f \circ g$

Reformulate flm's laws in terms of the $\diamond$ operation:

- flm's left and right identity laws: $\text{pure} \diamond f = f$ and $f \diamond \text{pure} = f$
- Associativity law: $(f \diamond g) \diamond h = f \diamond (g \diamond h)$
  - Follows from the flm law: $f \circ \text{flm}\,(g \circ \text{flm} h) = f \circ \text{flm}\, g \circ \text{flm}\, h$

* Motivation for categories and functors

Compare different "liftings" seen so far, and generalize

Category	Type $A \rightsquigarrow B$	Identity	Composition
plain functions	$A \Rightarrow B$	$\text{id} : A \Rightarrow A$	$f^{A \Rightarrow B} \circ g^{B \Rightarrow C}$
lifted to F	$F^A \Rightarrow F^B$	$\text{id} : F^A \Rightarrow F^A$	$f^{F^A \Rightarrow F^B} \circ g^{F^B \Rightarrow F^C}$
Kleisli over F	$A \Rightarrow F^B$	$\text{pure} : A \Rightarrow F^A$	$f^{A \Rightarrow F^B} \diamond g^{B \Rightarrow F^C}$

Category theory generalizes this situation

Category: a certain class of "twisted functions" $A \rightsquigarrow B$ called **morphisms**

- For any two morphisms $f^{A \rightsquigarrow B}$ and $g^{B \rightsquigarrow C}$ the **composition** morphism $f \diamond g$ of type $A \rightsquigarrow C$ must exist
- For each type A, the **identity** morphism $\text{id}_\diamond$ of type $A \rightsquigarrow A$ must exist
- Composition respects identity: $\text{id}_\diamond \diamond f = f$ and $f \diamond \text{id}_\diamond = f$
- Composition is associative: $(f \diamond g) \diamond h = f \diamond (g \diamond h)$

General **functor**: a map from one category to another

- A functor must `fmap` each morphism from one category to the other
- Functor laws: `fmap` must preserve identity and composition
  - What we call "functor" is called **endofunctor** in category theory
  - An endofunctor's `fmap` goes from plain functions to F-lifted functions

* From Kleisli back to `flatMap`

The Kleisli functions, $A \leadsto B \equiv A \Rightarrow S^B$, form a category iff S is a monad

- `fmap` and `flatMap` are computationally equivalent to Kleisli composition:
  - Define `flatMap` through Kleisli: $\text{flm}\, f^{A\Rightarrow S^B} \equiv \text{id}^{S^A \Rightarrow S^A} \diamond f$
  - Require two additional laws that connect $\diamond$, fmap, and $\circ$:
    - ★ Left naturality: $f^{A\Rightarrow B} \circ g^{B\Rightarrow S^C} = (f \circ \text{pure}) \diamond g$
    - ★ Right naturality: $f^{A\Rightarrow S^B} \circ \text{fmap}\, g^{B\Rightarrow C} = f \diamond (g \circ \text{pure})$
  - So, can define `fmap` through Kleisli: $\text{fmap}\, g^{A\Rightarrow B} \equiv \text{id}^{S^A\Rightarrow S^A} \diamond (g \circ \text{pure})$

The laws for `pure` and `flatMap` then follow from category axioms for Kleisli:

- Left and right identity laws follow from $\text{id} \diamond \text{pure} = \text{id}$ and $\text{pure} \diamond f = f$
- Associativity for `flatMap` follows from $(\text{id} \diamond f) \diamond g = \text{id} \diamond (f \diamond g)$
- Use "left naturality", get: $(f \circ g) \diamond h = (f \circ pure) \diamond g \diamond h = f \circ (g \diamond h)$
- Naturality for `pure`: $\text{pure} \circ \text{fmap}\, f = \text{pure} \diamond (f \circ \text{pure}) = f \circ \text{pure}$
- Define `flatten`: $\text{ftn} = \text{id}^{S^{S^A} \Rightarrow S^{S^A}} \diamond \text{id}^{S^A \Rightarrow S^A}$
- Naturality for `flatten`: $\text{ftn} \circ \text{fmap}\, f = \text{id} \diamond \text{id} \diamond (f \circ \text{pure}) = \text{id} \diamond \text{fmap}\, f$ and $\text{fmap}\,(\text{fmap}\, f) \circ \text{ftn} = \text{id} \diamond ((\text{fmap}\, f) \circ \text{pure}) \circ \text{id} \diamond \text{id} = \text{id} \diamond \text{fmap}\, f$

Structure of semigroups and monoids

- Semimonad contexts are combined associatively, as in a semigroup
  - A full monad includes an "empty" context, i.e. the identity element
  - Semigroup with an identity element is a monoid

Some constructions of semigroups and monoids (see code):

1. Any type Z is a semigroup with operation $z_1 \circledast z_2 = z_1$ (or z_2)
2. $1 + S$ is a monoid if S is (at least) a semigroup (or $S \equiv 0$)
3. List^A is a monoid (for any type A), also Seq^A etc.
4. The function type $A \Rightarrow A$ is a monoid (for any type A)
   - The operation $f \circledast g$ can be either $f \circ g$ or $g \circ f$
5. Any totally ordered type is a monoid, with $\circledast$ defined as max or min
6. $S_1 \times S_2$ is a semigroup (monoid) if S_1, S_2 are semigroups (monoids)
7. $S_1 + S_2$ is a semigroup (monoid) if S_1, S_2 are semigroups (monoids)
8. `M[S]` is a monoid if `M[_]` is a monad and `S` is a monoid
9. $S \times P$ is a semigroup if S is a semigroup that has an **action on** P
   - The "action" is $\alpha : S \Rightarrow P \Rightarrow P$ such that $\alpha(s_2) \circ \alpha(s_1) = \alpha(s_1 \circledast s_2)$
   - $S \times P$ is a "twisted product." Examples: $(A \Rightarrow A) \times A$; $\text{Bool} \times (1 + A)$

- Other examples of monoids: Int (many), String, Set^A, Akka's `Route`

Structure of (semi)monads

How to recognize a (semi)monad by its type? Open question!

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

Some constructions of exponential-polynomial (semi)monads:

1. $F^A \equiv Z$ (constant functor) for a fixed type Z
   - For a full monad, need to choose $Z = 1$
2. $F^A \equiv A \times G^A$ for any functor G^A (a full monad only if G^A is a monad)
3. $F^A \equiv G^A \times H^A$ for any (semi)monads G^A and H^A
   - but $G^A + H^A$ is generally *not* a semimonad
4. $F^A \equiv R \Rightarrow G^A$ is a (semi)monad for any (semi)monad G^A
5. $F^A \equiv A + G^A$ is a monad for a monad G^A (**free pointed** over G)
6. $F^A \equiv G^{Z+A\times W}$ is a monad if G is a monad and W a monoid
7. $F^A \equiv A + G^{F^A}$ (recursive) for any functor G^A (**free monad** over G)

   Semimonad-only constructions:
8. $F^A \equiv G^A + G^{F^A}$ (recursive) for any functor G^A
9. $F^A \equiv H^A \Rightarrow A \times G^A$ for any contrafunctor H^A and functor G^A
   - Obtain a full monad only when $G^A \equiv 1$, i.e. $F^A \equiv H^A \Rightarrow A$

Exercises II

1. Show that `M[S]` is a monoid if `M[_]` is a monad and `S` is a monoid.
2. A framework implements a "route" type R as $R \equiv Q \Rightarrow (E + S)$, where Q is a query, E is an error response, and S is a success response. A server is defined as a "sum" of several routes. For a given query Q, the response is the first route (if it exists) that yields a success. Implement the route "summation" operation and show that it makes R into a semigroup. What would be necessary to make R into a monoid?
3. Verify the associativity law for the semimonad $F^A \equiv Z + \text{Bool} \times A$.
4. Show that the functor $F^A \equiv \text{Boolean} \times M^A$ (where M^A is an arbitrary monad) can be made into a semimonad but not into a monad.
5. If W and R are arbitrary fixed types, which of the functors can be made into a semimonad: $F^A \equiv W \times (R \Rightarrow A)$, $G^A = R \Rightarrow (W \times A)$?
6. Show that $F^A \equiv (P \Rightarrow A) + (Q \Rightarrow A)$ is not a semimonad (cannot define `flatMap`) when P and Q are arbitrary, different types.
7. Implement the `flatten` and `pure` methods for $D^A \equiv 1 + A \times A$ (`type D[A] = Option[(A, A)]`) in at least two significantly different ways, and show that the monad laws always fail to hold. (D^A is not a monad!)

Exercises II (continued)

8. A programmer implemented the `fmap` method for $F^A \equiv A \times (A \Rightarrow Z)$ as

```
def fmap[A,B](f: A⇒B): ((A, A⇒Z)) ⇒ (B, B⇒Z) =
  { case (a, az) ⇒ (f(a), (_: B) ⇒ az(a)) }
```

   Show that this implementation fails to satisfy the functor laws.
9. Show that $P^A \equiv Z + W \times A$ is a (full) monad if W is a monoid.
10. Verify that the full monad laws hold for construction 4.
11. Implement `flatten` and `pure` for $F^A \equiv A + (R \Rightarrow A)$, where R is a fixed type, and show that all the monad laws hold.
12. For construction 5, show that an identity law would fail if `pure` were defined as `a ⇒ Right(Monad[G].pure(a))` instead of as `Left(a)`.
13. Implement the monad methods for $F^A \equiv (Z \Rightarrow 1 + A) \times \text{List}^A$ using the known monad constructions (no need to check the laws).
14. Implement the semimonad construction 2 by discarding the first effect (not the second), and show that the associativity law is still satisfied.
15. For semimonad construction 8, show that the associativity law holds.
16. Verify the identity laws for the State and Continuation monads.

Addendum: Miscellaneous remarks on monads

- A non-empty list $F^A \equiv A \times \text{List}^A$ is a semigroup but not a monoid.
- Any polynomial functor $F^A \equiv p(A)$ can be made into a monad when $p(x)$ is a polynomial of the form $p(x) = x^{n_1} + x^{n_2} + ... + x^{n_k}$ for some *positive* integers n_1, ..., n_k. Indeed, any F^A of this form may be built from the identity monad via constructions 3 and 5. To illustrate this, denote $E_1 \equiv 1$, $E_{n+1} \equiv 1 + E_n$. Monoid construction 2 makes E_n into monoids. Then the monads $E_n \Rightarrow A$ (reader) and $E_n \times A$ (writer) are equivalent to polynomial monads $A \times ... \times A$ and $A + ... + A$.
- Contrafunctors cannot be monads or semimonads: if H^A is a contrafunctor then H^{H^A} is a *functor*, so a natural transformation between H^{H^A} and H^A (in either direction) is impossible.
- An example of combining natural transformations: Given functors C, F, G and natural transformations $C^A \Rightarrow F^A$ and $C^A \Rightarrow G^A$ and taking the product, we get a natural transformation $C^A \Rightarrow F^A \times G^A$.
- If M^A is a monad then M^{M^A} is not automatically a monad (need counterexample?).
- Two monadic values $m_1, m_2 : M^A$ can be merged by ignoring the payload of one of them and merging the effects; and we can merge the effects in any chosen order:

  `for { x ← m₁; _ ← m₂ } yield x` or `for { _ ← m₁; x ← m₂ } yield x`
- A curious example: The functor $Q^A \equiv (A \Rightarrow Z) \Rightarrow 1 + A$ is not a monad (and not even a lawful applicative) but $M^A \equiv (A \Rightarrow 1 + 1) \Rightarrow 1 + A$ is a "search monad". More generally, a "selector monad" is $\left(A \Rightarrow P^1\right) \Rightarrow P^A$ for any functor P^A.