Chapter 7: Computations lifted to a functor context II

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 ← 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 = f(x)
z ← cont2(y)
```

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

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

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

- After `x ← c`, further computations will use *all those* `x`

```
x ← cont
y ← p(x)
z ← cont2(y)
```

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

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

Semimonad laws II: The laws for `flatMap`

To use the concise notation, denote `flatMap` by flm

A **semimonad** S^A has $\text{flm}^{[S,A,B]} : (A \Rightarrow S^B) \Rightarrow S^A \Rightarrow S^B$ 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. $\text{flm}\left(f^{A\Rightarrow S^B} \circ \text{flm}\, g^{B\Rightarrow S^C}\right) = \text{flm}\, f \circ \text{flm}\, g$ (composition)

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

Is there a shorter formulation of the laws?

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

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

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

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

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

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

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

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

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

Semimonad laws III: 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:

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. $\text{flm}\,(f \circ g_\uparrow) = \text{flm}\, f \circ g_\uparrow$ 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 for `flatten`
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 ftn's associativity: $\text{ftn}_\uparrow \circ \text{ftn} = \text{ftn} \circ \text{ftn}$

- `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 $F \leadsto G$
- Checking `flatten`'s associativity needs more work
  - The "accumulated context" is completely different for these monads

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

Semimonad laws IV: Checking the laws of `flatten`

- 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: $F^A \equiv (A \Rightarrow R) \Rightarrow R$;
    $\text{ftn} : ((((A \Rightarrow R) \Rightarrow R) \Rightarrow R) \Rightarrow R) \Rightarrow (A \Rightarrow R) \Rightarrow R$
- 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

Exercises II

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 (need semigroup for Z)
2. $F^A \equiv G^A \times H^A$ for any semimonads G^A and H^A
3. $F^A \equiv G^A + G^{F^A}$ (recursive) for any functor 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

(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: $f = \text{id}_{\diamond_{\textbf{Opt}}}$, so $f(a) = 0 + a$ and $\text{fmapOpt}_G f = \text{id}$
    - Hence we get $\text{fmapOpt}_F(\text{id}_{\diamond_{\textbf{Opt}}})(1 + a \times q) = 1 + a \times q$
- 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: $\text{id}_{\diamond_{\textbf{Opt}}}(x) \neq 1 + 0$, so $\text{fmapOpt}_F(\text{id}_{\diamond_{\textbf{Opt}}})(q + a \times p) = q + a \times p$
- Composition law:
  $(\text{fmapOpt}_F(f_1) \circ \text{fmapOpt}_F(f_2))(q + a \times p) = \text{fmapOpt}_F(f_1 \diamond_{\textbf{Opt}} f_2)(q + a \times p)$
- 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 $(f_1 \diamond_{\textbf{Opt}} f_2)(a) = 1 + 0$; to show $\text{fmapOpt}'_F(f_2)(\text{fmapOpt}'_F(f_1)(p)) = \text{fmapOpt}'_F(f_1 \diamond_{\textbf{Opt}} f_2)(p)$, use the inductive assumption about $\text{fmapOpt}'_F$ on p
- If $f_1(a) = 0 + b$ and $f_2(b) = 1 + 0$ then $(f_1 \diamond_{\textbf{Opt}} f_2)(a) = 1 + 0$; to show $\text{fmapOpt}_F(f_2)(0 + b \times \text{fmapOpt}'_F(f_1)(p)) = \text{fmapOpt}'_F(f_1 \diamond_{\textbf{Opt}} f_2)(p)$, 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 $F^A = R_1 \left[G^A + R_2 \left[H^A \right] \right]$
  - 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 $R_1 \left[G^A + R_2 \left[H^A \right] \right]$ 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 ⇒ 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).