Chapter 8: Applicative functors and profunctors
Part 2: Their laws and structure

Sergei Winitzki

Academy by the Bay

2018-07-01

Deriving the `ap` operation from `map2`

Can we avoid having to define map_n separately for each n?

- Use curried arguments, $\text{fmap}_2 : (A \Rightarrow B \Rightarrow Z) \Rightarrow F^A \Rightarrow F^B \Rightarrow F^Z$
- Set $A = B \Rightarrow Z$ and apply fmap_2 to the identity $\text{id}^{(B \Rightarrow Z) \Rightarrow (B \Rightarrow Z)}$: obtain $\text{ap}^{[B,Z]} : F^{B \Rightarrow Z} \Rightarrow F^B \Rightarrow F^Z \equiv \text{fmap}_2\,(\text{id})$
- The functions `fmap2` and `ap` are computationally equivalent:

$$\text{fmap}_2\,f^{A \Rightarrow B \Rightarrow Z} = \text{fmap}\,f \circ \text{ap}$$

$$F^A \xrightarrow[\text{fmap}_2\,(f^{A \Rightarrow B \Rightarrow Z})]{\quad\xrightarrow{\text{fmap}\,f}\; F^{B \Rightarrow Z} \;\xrightarrow{\text{ap}}\quad} \left(F^B \Rightarrow F^Z\right)$$

- The functions `fmap3`, `fmap4` etc. can be defined similarly:

$$\text{fmap}_3\,f^{A \Rightarrow B \Rightarrow C \Rightarrow Z} = \text{fmap}\,f \circ \text{ap} \circ \text{fmap}_{F^B \Rightarrow ?}\text{ap}$$

$$F^A \xrightarrow[\text{fmap}_3\,(f^{A \Rightarrow B \Rightarrow C \Rightarrow Z})]{\quad\xrightarrow{\text{fmap}\,f}\; F^{B \Rightarrow C \Rightarrow Z} \xrightarrow{\text{ap}^{[B,C \Rightarrow Z]}} \left(F^B \Rightarrow F^{C \Rightarrow Z}\right) \xrightarrow{\text{fmap}_{F^B \Rightarrow ?}\text{ap}^{[C,Z]}}\quad} \left(F^B \Rightarrow F^C \Rightarrow F^Z\right)$$

- Using the infix syntax will get rid of $\text{fmap}_{F^B \Rightarrow ?}\text{ap}$ (see example code)
  - Note the pattern: a natural transformation is equivalent to a lifting

Deriving the `zip` operation from `map2`

- Note: Function types $A \Rightarrow B \Rightarrow C$ and $A \times B \Rightarrow C$ are equivalent
- Uncurry fmap_2 to fmap2 : $(A \times B \Rightarrow C) \Rightarrow F^A \times F^B \Rightarrow F^C$
- Compute $\text{fmap2}(f)$ with $f = \text{id}^{A \times B \Rightarrow A \times B}$, expecting to obtain a simpler natural transformation:

$$\text{zip} : F^A \times F^B \Rightarrow F^{A \times B}$$

- This is quite similar to `zip` for lists:
  `List(1, 2).zip(List(10, 20)) = List((1, 10), (2, 20))`
- The functions `zip` and `fmap2` are computationally equivalent:

$$\text{zip} = \text{fmap2}(\text{id})$$

$$\text{fmap2}(f^{A \times B \Rightarrow C}) = \text{zip} \circ \text{fmap}\, f$$

- The functor F is **zippable** if such a `zip` exists (with appropriate laws)
  - The same pattern: a natural transformation is equivalent to a lifting

* Equivalence of the operations `ap` and `zip`

- Set $A \equiv B \Rightarrow C$, get $\text{zip}^{[B \Rightarrow C, B]} : F^{B \Rightarrow C} \times F^B \Rightarrow F^{(B \Rightarrow C) \times B}$
- Use `eval` $: (B \Rightarrow C) \times B \Rightarrow C$ and $\text{fmap}\,(\text{eval}) : F^{(B \Rightarrow C) \times B} \Rightarrow F^C$
- Uncurry: $\text{app}^{[B,C]} : F^{B \Rightarrow C} \times F^B \Rightarrow F^C \equiv \text{zip} \circ \text{fmap}\,(\text{eval})$
- The functions `zip` and `app` are computationally equivalent:
  - use pair $: (A \Rightarrow B \Rightarrow A \times B) = a^A \Rightarrow b^B \Rightarrow a \times b$
  - use $\text{fmap}\,(\text{pair}) \equiv \text{pair}^\uparrow$ on an fa^{F^A}, get $(\text{pair}^\uparrow fa) : F^{B \Rightarrow A \times B}$; then

$$\text{zip}\,(fa \times fb) = \text{app}\left((\text{pair}^\uparrow fa) \times fb\right)$$

$$\text{app}^{[B \Rightarrow C, B]} = \text{zip}^{[B \Rightarrow C, B]} \circ \text{fmap}\,(\text{eval})$$

- Rewrite this using curried arguments: $\text{fzip}^{[A,B]} : F^A \Rightarrow F^B \Rightarrow F^{A \times B}$; $\text{ap}^{[B,C]} : F^{B \Rightarrow C} \Rightarrow F^B \Rightarrow F^C$; then $\text{ap}\,f = \text{fzip}\,f \circ \text{fmap}\,(\text{eval})$.
- Now $\text{fzip}\,p^{F^A} q^{F^B} = \text{ap}\,(\text{pair}^\uparrow p)\,q$, hence we may omit the argument q: $\text{fzip} = \text{pair}^\uparrow \circ \text{ap}$. With explicit types: $\text{fzip}^{[A,B]} = \text{pair}^\uparrow \circ \text{ap}^{[B, A \Rightarrow B]}$.

Treat `map2` as a replacement for a monadic block with independent effects:

```
for {                              map2 (
  x ← cont1                          cont1,
  y ← cont2                          cont2
} yield g(x, y)                    ) { (x, y) ⇒ g(x, y) }
```

- Main idea: Formulate the monad laws in terms of `map2` and `pure`

Naturality laws: Manipulate data in one of the containers

```
for {                              for {
  x ← cont1.map(f)                   x ← cont1
  y ← cont2                          y ← cont2
} yield g(x, y)                    } yield g(f(x), y)
```

and similarly for `cont2` instead of `cont1`; now rewrite in terms of for `map2`:

- **Left naturality** for `map2`:

  ```
  map2(cont1.map(f), cont2)(g)
    = map2(cont1, cont2){ (x, y) ⇒ g(f(x), y) }
  ```

- **Right naturality** for `map2`:

  ```
  map2(cont1, cont2.map(f))(g)
    = map2(cont1, cont2){ (x, y) ⇒ g(x, f(y)) }
  ```

Associativity and identity laws for `map2`

Inline two generators out of three, in two different ways:

```
for {                             for {
  x ← cont1                         (x, y) ← for {
  (y, z) ← for {                             xx ← cont1
         yy ← cont2                          yy ← cont2
         zz ← cont3                        } yield (xx, yy)
       } yield (yy, zz)             z ← cont3
} yield g(x, y, z)                } yield g(x, y, z)
```

Write this in terms of `map2` to obtain the **associativity law** for `map2`:

```
map2(cont1, map2(cont2, cont3)((_,_)){ case(x,(y,z))⇒g(x,y,z)}
   = map2(map2(cont1, cont2)((_,_)), cont3){ case((x,y),z))⇒g(x,y,z)}
```

Empty context precedes a generator, or follows a generator:

```
for { x ← pure(a)                 for {
      y ← cont                      y ← cont
} yield g(x, y)                   } yield g(a, y)
```

Write this in terms of `map2` to obtain the **identity laws** for `map2` and `pure`:

```
map2(pure(a), cont)(g) = cont.map { y ⇒ g(a, y) }
map2(cont, pure(b))(g) = cont.map { x ⇒ g(x, b) }
```

Deriving the laws for `zip`: naturality

- The laws for `map2` in a short notation; here $f \otimes g \equiv \{a \times b \Rightarrow f(a) \times g(b)\}$

$$\mathsf{fmap2}\left(g^{A \times B \Rightarrow C}\right)\left(f^{\uparrow} q_1 \times q_2\right) = \mathsf{fmap2}\left((f \otimes \mathsf{id}) \circ g\right)(q_1 \times q_2)$$

$$\mathsf{fmap2}\left(g^{A \times B \Rightarrow C}\right)\left(q_1 \times f^{\uparrow} q_2\right) = \mathsf{fmap2}\left((\mathsf{id} \otimes f) \circ g\right)(q_1 \times q_2)$$

$$\mathsf{fmap2}\left(g_{1.23}\right)(q_1 \times \mathsf{fmap2}\,(\mathsf{id})\,(q_2 \times q_3)) = \mathsf{fmap2}\,(g_{12.3})\,(\mathsf{fmap2}\,(\mathsf{id})\,(q_1 \times q_2) \times q_3)$$

$$\mathsf{fmap2}\left(g^{A \times B \Rightarrow C}\right)\left(\mathsf{pure}\,a^A \times q_2^{F^B}\right) = (b \Rightarrow g\,(a \times b))^{\uparrow} q_2$$

$$\mathsf{fmap2}\left(g^{A \times B \Rightarrow C}\right)\left(q_1^{F^A} \times \mathsf{pure}\,b^B\right) = (a \Rightarrow g\,(a \times b))^{\uparrow} q_1$$

- Express `map2` through `zip`:

$$\mathsf{fmap_2}\,g^{A \times B \Rightarrow C}\left(q_1^{F^A} \times q_2^{F^B}\right) \equiv \left(\mathsf{zip} \circ g^{\uparrow}\right)(q_1 \times q_2)$$

$$\mathsf{fmap_2}\,g^{A \times B \Rightarrow C} \equiv \mathsf{zip} \circ g^{\uparrow}$$

- Combine the two naturality laws into one by using two functions f_1, f_2:

$$\left(f_1^{\uparrow} \otimes f_2^{\uparrow}\right) \circ \mathsf{fmap2}\,g = \mathsf{fmap2}\left((f_1 \otimes f_2)^{\uparrow} \circ g\right)$$

$$\left(f_1^{\uparrow} \otimes f_2^{\uparrow}\right) \circ \mathsf{zip} \circ g^{\uparrow} = \mathsf{zip} \circ (f_1 \otimes f_2)^{\uparrow} \circ g^{\uparrow}$$

- The **naturality law** for `zip` then becomes: $\left(f_1^{\uparrow} \otimes f_2^{\uparrow}\right) \circ \mathsf{zip} = \mathsf{zip} \circ (f_1 \otimes f_2)^{\uparrow}$

Deriving the laws for `zip`: associativity

- Express `map2` through `zip` and substitute into the associativity law:
$$g^{\uparrow}_{1.23}\left(\mathrm{zip}\left(q_1 \times \mathrm{zip}\left(q_2 \times q_3\right)\right)\right) = g^{\uparrow}_{12.3}\left(\mathrm{zip}\left(\mathrm{zip}\left(q_1 \times q_2\right) \times q_3\right)\right)$$

- The arbitrary function g is preceded by transformations of the tuples,
$$a \times (b \times c) \equiv (a \times b) \times c \quad \text{(type isomorphism)}$$

- Assume that the isomorphism transformations are applied as needed, then we may formulate the **associativity law** for `zip` more concisely:
$$\mathrm{zip}\left(q_1 \times \mathrm{zip}\left(q_2 \times q_3\right)\right) \cong \mathrm{zip}\left(\mathrm{zip}\left(q_1 \times q_2\right) \times q_3\right)$$
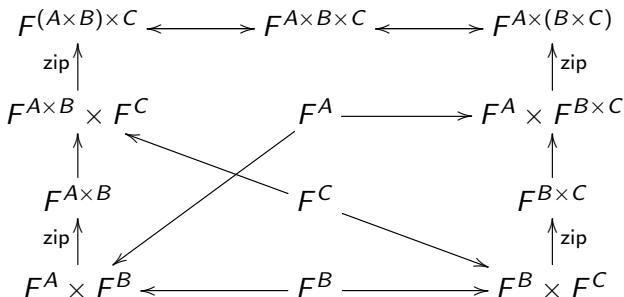
Deriving the laws for `zip`: identity laws

- Identity laws seem to be complicated, e.g. the left identity:
$$g^{\uparrow}\left(\mathrm{zip}\left(\mathrm{pure}\, a \times q\right)\right) = \left(b \Rightarrow g\left(a \times b\right)\right)^{\uparrow} q$$

- Replace `pure` by a simpler "wrapped unit" method `unit: F[Unit]`
$$\mathrm{unit}^{F^{\mathbf{1}}} \equiv \mathrm{pure}\,(1)\,; \quad \mathrm{pure}(a^A) = (1 \Rightarrow a)^{\uparrow}\,\mathrm{unit}$$

  Then the left identity law can be simplified using left naturality:
$$g^{\uparrow}\left(\mathrm{zip}\left(\left((1 \Rightarrow a)^{\uparrow}\,\mathrm{unit}\right) \times q\right)\right) = g^{\uparrow}\left(((1 \Rightarrow a) \times \mathrm{id})^{\uparrow}\,\mathrm{zip}\,(\mathrm{unit} \times q)\right)$$

- Denote $\phi^{B \Rightarrow 1 \times B} \equiv b \Rightarrow 1 \times b$ and $\beta_a^{1 \times B \Rightarrow A \times B} \equiv (1 \Rightarrow a) \times \mathrm{id}$; then the function $b \Rightarrow g\,(a \times b)$ can be expressed more simply as $\phi \circ \beta_a \circ g$, and the naturality law becomes
$$g^{\uparrow}(\beta_a^{\uparrow}\,\mathrm{zip}\,(\mathrm{unit} \times q)) = (\beta_a \circ g)^{\uparrow}\,(\mathrm{zip}\,(\mathrm{unit} \times q)) = (\phi \circ \beta_a \circ g)^{\uparrow} q = (\beta_a \circ g)^{\uparrow}\,(\phi^{\uparrow} q)$$

  Omitting the common prefix $(\beta_a \circ g)^{\uparrow}$, we obtain the **left identity** law:
$$\mathrm{zip}\,(\mathrm{unit} \times q) = \phi^{\uparrow} q$$

  - Note that $\phi^{\uparrow}$ is an isomorphism between F^B and $F^{1 \times B}$
  - Assume that this isomorphism is applied as needed, then we may write
$$\mathrm{zip}\,(\mathrm{unit} \times q) \cong q$$

Applicative laws as monoid laws

- Use infix syntax for `zip` and write $\mathrm{zip}\,(p \times q) \equiv p \bowtie q$
- Then the associativity and identity laws may be written as

$$q_1 \bowtie (q_2 \bowtie q_3) \cong (q_1 \bowtie q_2) \bowtie q_3$$
$$(\mathrm{unit} \bowtie q) \cong q$$
$$(q \bowtie \mathrm{unit}) \cong q$$

  These are the laws of a monoid (with some assumed transformations)
- Naturality law for `zip` written in the infix syntax:

$$f_1^{\uparrow} q_1 \bowtie f_2^{\uparrow} q_2 = (f_1 \otimes f_2)^{\uparrow} (q_1 \bowtie q_2)$$

- `unit` has no laws; the naturality for `pure` follows automatically
- The laws are simplest when formulated in terms of `zip` and `unit`
  - ▶ Naturality for `zip` will usually follow from parametricity
- "Zippable" functors have only the associativity and naturality laws
- Applicative functors are a strict subset of monadic functors
  - ▶ There are applicative functors that cannot be monads
  - ▶ Applicative functor implementation may disagree with the monad

Constructions of applicative functors

- All monadic constructions still hold for applicative functors
- Additionally, there are some non-monadic constructions

1. $F^A \equiv 1$ (constant functor) and $F^A \equiv A$ (identity functor)
2. $F^A \equiv G^A \times H^A$ for any applicative G^A and H^A
   - but $G^A + H^A$ is in general *not* applicative
3. $F^A \equiv R \Rightarrow G^A$ for any applicative G^A and any fixed type R
4. $F^A \equiv A + G^A$ for any applicative G^A (**free pointed** over G)
5. $F^A \equiv A + G^{F^A}$ (recursive) for any functor G^A (**free monad** over G)
6. $F^A \equiv H^A \Rightarrow A$ for any contrafunctor H^A

   Constructions that are not monadic:
7. $F^A \equiv Z$ (constant functor, Z a monoid)
8. $F^A \equiv Z + A \times W$ when W and Z are monoids (nontrivial instance)
9. $F^A \equiv G^{H^A}$ when both G and H are applicative
10. $F^A \equiv G^A + H^{G^A}$ where H is any functor and G is applicative

All non-parameterized exp-poly types are monoids

- Known monoid constructions (Chapter 7) implement $X + Y$, $X \times Y$, $X \Rightarrow Y$ as monoids when X and Y are monoids
- All primitive types have at least one monoid instance
  - `Int`, `Float`, `Double`, `Char`, `Boolean` are "numeric" monoids
  - `Unit` is a trivial monoid
  - `Seq[A]`, `Set[A]`, `Map[K,V]` are set-like monoids
  - `String` is equivalent to a sequence of integers
- Therefore, all exponential-polynomial types without type parameters are monoids in at least one way
- Example of an exp-poly type without type parameters:
  $\text{Int} + \text{String} \times \text{String} \times (\text{Int} \Rightarrow \text{Bool}) + (\text{Bool} \times \text{String} \Rightarrow 1 + \text{String})$
- Example of a type with parameters, not a monoid: $A \Rightarrow B$

Therefore, all non-parameterized polynomial functors are applicative

- $F^A = 1 + A \times A$ (this is not a monad!)
- $F^A = 1 + A \times A \times Z$ where Z is a monoid

Definition and constructions of applicative contrafunctors

- Express

- Express

Definition and constructions of applicative profunctors

- Express

Exercises

1. Show that $F^A \equiv (Z \Rightarrow A) \Rightarrow 1 + A$ is not applicative.