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:

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

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

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

- 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 fmap2 (f) with $f = \mathrm{id}^{A \times B \Rightarrow A \times B}$, expecting to obtain a simpler natural transformation:

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

- 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:

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

- 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: $\text{fzip}^{[A,B]} = \text{pair}^{\uparrow} \circ \text{ap}^{[B, A \Rightarrow B]}$.

Motivation for applicative laws. Naturality laws for `map2`

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 $f \otimes g \equiv \{a \times b \Rightarrow f(a) \times g(b)\}$

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

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

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

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

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

- Express `map2` through `zip`:

$$\text{fmap}_2\, g^{A \times B \Rightarrow C}\left(q_1^{F^A} \times q_2^{F^B}\right) \equiv \left(\text{zip} \circ g^{\uparrow}\right)(q_1 \times q_2)$$

$$\text{fmap}_2\, g^{A \times B \Rightarrow C} \equiv \text{zip} \circ g^{\uparrow}$$

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

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

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

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

Deriving the laws for `zip`: associativity

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

- 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:
$$\text{zip}\left(q_1 \times \text{zip}\left(q_2 \times q_3\right)\right) \cong \text{zip}\left(\text{zip}\left(q_1 \times q_2\right) \times q_3\right)$$
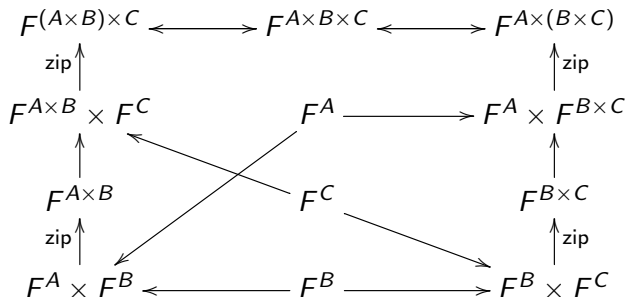
Deriving the laws for `zip`: identity laws

- Identity laws seem to be complicated, e.g. the left identity:

$$g^\uparrow (\text{zip} (\text{pure } a \times q)) = (b \Rightarrow g (a \times b))^\uparrow q$$

- Replace `pure` by a simpler "wrapped unit" method `unit: F[Unit]`

$$\text{unit}^{F^1} \equiv \text{pure} (1) ; \quad \text{pure}(a^A) = (1 \Rightarrow a)^\uparrow \text{unit}$$

Then the left identity law can be simplified using left naturality:

$$g^\uparrow \left(\text{zip} \left(\left((1 \Rightarrow a)^\uparrow \text{unit} \right) \times q \right) \right) = g^\uparrow \left(((1 \Rightarrow a) \times \text{id})^\uparrow \text{zip} (\text{unit} \times q) \right)$$

- Denote $\phi^{B \Rightarrow 1 \times B} \equiv b \Rightarrow 1 \times b$ and $\beta_a^{1 \times B \Rightarrow A \times B} \equiv (1 \Rightarrow a) \times \text{id}$; 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

$$g^\uparrow (\beta_a^\uparrow \text{zip} (\text{unit} \times q)) = (\beta_a \circ g)^\uparrow (\text{zip} (\text{unit} \times q)) = (\phi \circ \beta_a \circ g)^\uparrow q = (\beta_a \circ g)^\uparrow (\phi^\uparrow q)$$

Omitting the common prefix $(\beta_a \circ g)^\uparrow$, we obtain the **left identity** law:

$$\text{zip} (\text{unit} \times q) = \phi^\uparrow q$$

  - 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

$$\text{zip} (\text{unit} \times q) \cong q$$

Applicative laws as monoid laws

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

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

  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. $F^A \equiv A + G^A$ for any applicative G^A (**free pointed** over G)
4. $F^A \equiv A + G^{F^A}$ (recursive) for any functor G^A (**free monad** over G)
5. $F^A \equiv H^A \Rightarrow A$ for any contrafunctor H^A
   Constructions that are not monadic:
6. $F^A \equiv Z$ (constant functor, Z a monoid)
7. $F^A \equiv Z + G^A$ for any applicative G^A and monoid Z
8. $F^A \equiv G^{H^A}$ when both G and H are applicative
9. $F^A \equiv G^A + H^{G^A}$ where H is any functor and G is applicative

All non-parameterized exp-poly types are monoids

- Using known monoid constructions (Chapter 7), we can 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
  - `Seq[A]`, `Set[A]`, `Map[K,V]` are set-like monoids
  - `String` is equivalent to a sequence of integers; `Unit` is a trivial monoid
- Therefore, all exponential-polynomial types without type parameters are monoids in at least one way
- Example of an exponential-polynomial 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, which is not a monoid: $A \Rightarrow B$

By constructions 1, 3, and 7, *all* polynomial F^A with monoidal parameters are applicative: write $F^A = Z_1 + A \times (Z_2 + A \times ...)$ with some monoids Z_i

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

Examples of non-polynomial functors that are not applicative:

- $F^A \equiv (A \Rightarrow R) \Rightarrow S; \quad F^A \equiv (R \Rightarrow A) + (S \Rightarrow A)$

Definition and constructions of applicative contrafunctors

- The applicative functor laws, if formulated via `zip` and `unit`, do not use `map` and therefore can be used for contrafunctors
- Define an **applicative contrafunctor** C^A as having `zip` and `unit`:

$$\text{zip} : C^A \times C^B \Rightarrow C^{A \times B}; \quad \text{unit} : C^1$$

- Identity and associativity laws must hold for `zip` and `unit`
  - Note: applying `contramap` to the function $a \times b \Rightarrow a$ will yield some $C^A \Rightarrow C^{A \times B}$, but this will not give a valid implementation of `zip`!
- Naturality must hold for `zip`, but with `contramap` instead of `map`

Applicative contrafunctor constructions:

1. $C^A \equiv Z$ (constant functor, Z a monoid)
2. $C^A \equiv G^A \times H^A$ for any applicative contrafunctors G^A and H^A
3. $C^A \equiv G^A + H^A$ for any applicative contrafunctors G^A and H^A
4. $C^A \equiv H^A \Rightarrow G^A$ for any functor H^A and applicative contrafunctor G^A
5. $C^A \equiv H^{G^A}$ for any functor H^A and applicative contrafunctor G^A

- *All* exponential-polynomial contrafunctors with monoidal parameters are applicative! (These constructions cover all exp-poly cases.)

Definition and constructions of applicative profunctors

- **Profunctors** have the type parameter in both covariant and contravariant positions; they are neither functors nor contrafunctors
- Examples of profunctors: $P^A \equiv \text{Int} \times A \Rightarrow A$; $\quad P^A \equiv A + (A \Rightarrow R)$
- *All* exp-poly type constructors are profunctors since the type parameter is always in either a covariant or a contravariant position
- Definition of **applicative profunctor**: has `zip` and `unit` with the laws

Applicative profunctors have all previous constructions, and additionally:

1. $P^A \equiv G^A \times H^A$ for any applicative profunctors G^A and H^A
2. $P^A \equiv Z + G^A$ for any applicative profunctor G^A and monoid Z
3. $P^A \equiv A + G^A$ for any applicative profunctor G^A
4. $P^A \equiv G^A + H^{G^A}$ for any functor H^A and applicative profunctor G^A
5. $P^A \equiv H^A \Rightarrow A$ for any profunctor H^A
6. $P^A \equiv H^{G^A}$ and G^{H^A} for any functor H^A and applicative profunctor G^A

Examples of non-applicative profunctors:

- $P^A \equiv (A \Rightarrow A) + (R \Rightarrow A)$; $\quad P^A \equiv (A \Rightarrow A) \Rightarrow 1 + A$

Exercises

1. Show that $F^A \equiv (A \Rightarrow Z) \Rightarrow (1 + A)$ is a functor but not applicative.
2. Show that P^S is a monoid if S is a monoid and P is any applicative functor, contrafunctor, or profunctor.