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 $A \equiv (B \Rightarrow Z)$ 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`

- The types $A \Rightarrow B \Rightarrow C$ and $A \times B \Rightarrow C$ are equivalent (curry/uncurry)
- Uncurry fmap$_2$ to fmap2 : $(A \times B \Rightarrow C) \Rightarrow F^A \times F^B \Rightarrow F^C$
- Compute 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

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

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

- 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 law

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

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

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

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

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

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

- 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 law

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

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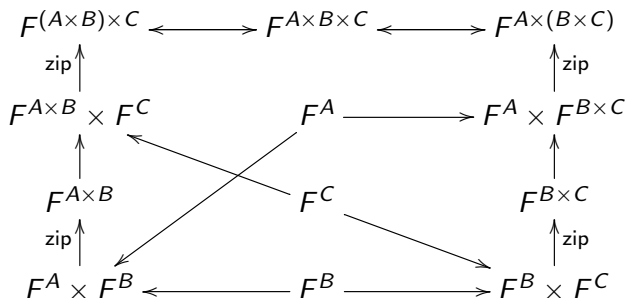
Deriving the laws for `zip`: identity laws

- 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 an *equivalent* "wrapped unit" method `wu: F[Unit]`

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

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

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

- Denote $\phi^{B \Rightarrow \mathbf{1} \times B} \equiv b \Rightarrow 1 \times b$ and $\beta_a^{\mathbf{1} \times B \Rightarrow A \times B} \equiv (1 \Rightarrow a) \otimes \text{id}$; then the function $b \Rightarrow g\,(a \times b)$ can be expressed more simply as $\phi \circ \beta_a \circ g$, and the identity law becomes

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

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

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

- ▸ Note that $\phi^\uparrow$ is an isomorphism between F^B and $F^{\mathbf{1} \times B}$
    - ★ Assume that this isomorphism is applied as needed, then we may write

$$\text{zip}\,(\text{wu} \times q) \cong q$$

- ▸ Similarly, the **right identity** law can be written as $\text{zip}\,(q \times \text{wu}) \cong q$

Similarity between applicative laws and monoid laws

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

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

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

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

- `wu` has no laws; the naturality for `pure` follows automatically
- The laws are simplest when formulated in terms of `zip` and `wu`
  - Naturality for `zip` will usually follow from parametricity
    - A third naturality law for `map2` follows from defining `map2` through `zip`!
- "Zippable" functors have only the associativity and naturality laws
- Applicative functors are a strict superset of monadic functors
  - There are applicative functors that *cannot* be monads
  - Applicative functor implementation may disagree with the monad

A third naturality law for `map2`

- There must be one more naturality law for `map2`
- Transform the result of a `map2`:

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

- Write this in terms of `map2`, obtain a third naturality law:

```
map2(cont1, cont2)(g).map(f)
    = map2(cont1, cont2)(g andThen f)
```

$$\mathrm{fmap2}\,(g) \circ f^{\uparrow} = \mathrm{fmap2}\,(g \circ f)$$

$$f^{\uparrow}\,(\mathrm{fmap2}\,(g)\,(p \times q)) = \mathrm{fmap2}\,(g \circ f)\,(p \times q)$$

- This law automatically follows if we define `map2` through `zip`:

$$\mathrm{fmap2}\,(g) \circ f^{\uparrow} = \mathrm{zip} \circ g^{\uparrow} \circ f^{\uparrow} = \mathrm{zip} \circ (g \circ f)^{\uparrow}$$

- Note: We always have one naturality law per type parameter

Applicative operation `ap` as a "lifting"

- Consider `ap` as a "lifting" since it has type $F^{A \Rightarrow B} \Rightarrow (F^A \Rightarrow F^B)$
- A "lifting" should obey the identity and the composition laws
  - ▸ An "identity" value of type $F^{A \Rightarrow A}$, mapped to $\mathrm{id}^{F^A \Rightarrow F^A}$ by `ap`
    - ★ A good candidate for that value is $\mathrm{id}_{\odot} \equiv \mathrm{pure}\,(\mathrm{id}^{A \Rightarrow A})$
  - ▸ A "composition" of an $F^{A \Rightarrow B}$ and an $F^{B \Rightarrow C}$, yielding an $F^{A \Rightarrow C}$
    - ★ We can use `map2` to implement this composition, denoted $g \odot h$:

$$g^{F^{A \Rightarrow B}} \odot h^{F^{B \Rightarrow C}} \equiv \mathsf{fmap2}\,(p^{A \Rightarrow B} \times q^{B \Rightarrow C} \Rightarrow p \circ q)\,(g, h)$$

- What are the laws that follow for $g \odot h$ from the `map2` laws?

$$\mathrm{id}_{\odot} \odot h = h; \quad g \odot \mathrm{id}_{\odot} = g$$

$$g^{F^{A \Rightarrow B}} \odot (h^{F^{B \Rightarrow C}} \odot k^{F^{C \Rightarrow D}}) = (g \odot h) \odot k$$

$$\left((x^{B \Rightarrow C} \Rightarrow f^{A \Rightarrow B} \circ x)^{\uparrow} g^{F^{B \Rightarrow C}} \right) \odot h^{F^{C \Rightarrow D}} = (x^{B \Rightarrow D} \Rightarrow f^{A \Rightarrow B} \circ x)^{\uparrow}\,(g \odot h)$$

$$g^{F^{A \Rightarrow B}} \odot \left((x^{B \Rightarrow C} \Rightarrow x \circ f^{C \Rightarrow D})^{\uparrow} h^{F^{B \Rightarrow C}} \right) = (x^{A \Rightarrow C} \Rightarrow x \circ f^{C \Rightarrow D})^{\uparrow}\,(g \odot h)$$

  - ▸ The first 3 laws are the identity & associativity laws of a *category*
    - ★ The morphism type is $A \rightsquigarrow B \equiv F^{A \Rightarrow B}$, the composition is $\odot$
  - ▸ The last 2 laws are naturality laws, connecting fmap and $\odot$
- Therefore `ap` is a functor's "lifting" of morphisms from two categories

Deriving the category laws for $(\text{id}_\odot, \odot)$

The five laws for $\text{id}_\odot$ and $\odot$ follow from the five `map2` laws

- Consider $\text{id}_\odot \odot h$ and substitute the definition of $\odot$ via `map2`, cf. slide 7:
  $\text{id}_\odot \odot h = \text{fmap2}\,(p \times q \Rightarrow p \circ q)\,(\text{pure}\,(\text{id}) \times h) = (b \Rightarrow \text{id} \circ b)^\uparrow h = h$
- The law $g \odot \text{id}_\odot = g$ is derived similarly
- Associativity law: $g \odot (h \odot k) = \text{fmap2}\,(\circ)\,(g \times \text{fmap2}\,(\circ)\,(h \times k))$ The 3rd naturality law gives: $\text{fmap2}\,(\circ)\,(h \times k) = (\circ)^\uparrow\,(\text{fmap2}\,(\text{id})\,(h \times k))$, and then:

$$g \odot (h \odot k) = \text{fmap2}\,(x \times (y \times z) \Rightarrow x \circ y \circ z)\,(g \times \text{fmap2}\,(\text{id})\,(h \times k))$$
$$(g \odot h) \odot k = \text{fmap2}\,((x \times y) \times z \Rightarrow x \circ y \circ z)\,(\text{fmap2}\,(\text{id})\,(g \times h) \times k)$$

  Now the associativity law for `fmap2` yields $g \odot (h \odot k) = (g \odot h) \odot k$
- Derive naturality laws for $\odot$ from the three map_2 naturality laws:
  $((x \Rightarrow f \circ x)^\uparrow g) \odot h = \text{fmap2}\,(\circ)\,((x \Rightarrow f \circ x)^\uparrow g \times h) =$
  $\text{fmap2}\,(x \times y \Rightarrow f \circ x \circ y)\,(g \times h) = (x \Rightarrow f \circ x)^\uparrow\,(\text{fmap2}\,(\circ)\,(g \times h)) =$
  $(x \Rightarrow f \circ x)^\uparrow\,(g \odot h)$
- The law is $g \odot (x \Rightarrow x \circ f)^\uparrow h = (x \Rightarrow x \circ f)^\uparrow\,(g \odot h)$ is derived similarly

Deriving the functor laws for `ap`

Now that we established the laws for $\odot$, we have `ap` laws:

$$\mathsf{ap}^{[B,Z]} : F^{B \Rightarrow Z} \Rightarrow F^B \Rightarrow F^Z = \mathsf{fmap}_2 \left(\mathsf{id}^{(B \Rightarrow Z) \Rightarrow (B \Rightarrow Z)} \right)$$

Identity law: $\mathsf{ap}(\mathsf{id}_\odot) = \mathsf{id}^{F^A \Rightarrow F^A}$

- Derivation: $\mathsf{ap}(\mathsf{id}_\odot^{F^{A \Rightarrow A}})(q^{F^A}) = \mathsf{fmap}_2(\mathsf{id}^{(A \Rightarrow A) \Rightarrow A \Rightarrow A})(\mathsf{pure}(\mathsf{id}^{A \Rightarrow A}))(q^{F^A}) =$
  $\mathsf{fmap2}\,(f \times x \Rightarrow f(x))\,(\mathsf{pure}(\mathsf{id}) \times q) = (x \Rightarrow \mathsf{id}(x))^\uparrow\,q = \mathsf{id}^\uparrow q = q$
- Easier derivation: first, express `ap` via $\odot$ using the isomorphisms

$$A \cong 1 \Rightarrow A; \quad F^A \cong F^{1 \Rightarrow A}$$

  Then $\mathsf{ap}(p^{F^{B \Rightarrow Z}})(q^{F^B}) \cong q^{F^{1 \Rightarrow B}} \odot p^{F^{B \Rightarrow Z}}$ and so $\mathsf{ap}(\mathsf{id}_\odot)(q) \cong q \odot \mathsf{id}_\odot = q$

Composition law: $\mathsf{ap}(g \odot h) = \mathsf{ap}(g) \circ \mathsf{ap}(h)$

- Derivation: use $\mathsf{ap}\,p\,q \cong q \odot p$ to get $\mathsf{ap}(g \odot h)(q) \cong q \odot (g \odot h)$ while
  $(\mathsf{ap}(g) \circ \mathsf{ap}(h))\,q = \mathsf{ap}(h)(\mathsf{ap}(g)(q)) \cong \mathsf{ap}(h)(q \odot g) \cong (q \odot g) \odot h$

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 do not correspond to monadic ones:
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

- Applicative that disagrees with its monad: $F^A \equiv 1 + (1 \Rightarrow A \times F^A)$
- Examples of non-applicative functors: $F^A \equiv (P \Rightarrow A) + (Q \Rightarrow A)$, $F^A \equiv (A \Rightarrow P) \Rightarrow Q$, $F^A \equiv (A \Rightarrow P) \Rightarrow 1 + A$

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:
  Int + String × String × (Int ⇒ Bool) + (Bool × String ⇒ 1 + String)
- Example of a non-monoid type with type parameters: $A \Rightarrow B$

By constructions 1, 2, 6, 7, *all* polynomial F^A with monoidal coefficients are applicative: write $F^A = Z_1 + A \times (Z_2 + A \times ...)$ with some monoids Z_i

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

Previous examples of non-applicative functors are all *non-polynomial*

Definition and constructions of applicative contrafunctors

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

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

- Identity and associativity laws must hold for `zip` and `wu`
  - 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`
  - There are no corresponding `pure` or `contraap`! But have $\forall A : P^A$

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 coefficients are applicative! (These constructions cover all exp-poly cases.)

Definition and constructions of applicative profunctors

- **Profunctors** have the type parameter in both contravariant and covariant positions; they can have neither `map` nor `contramap`
  - They have `dimap` of type $(A \Rightarrow B) \times (B \Rightarrow A) \Rightarrow (F^A \Rightarrow F^B)$
- Examples of profunctors: $P^A \equiv \text{Int} \times A \Rightarrow A$; $P^A \equiv A + (A \Rightarrow R)$
- Example of non-profunctor: a GADT, $F^A \equiv \text{String}^{F^{\textbf{Int}}} + \text{Int}^{F^{\textbf{1}}}$
- *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 `wu` with the laws
  - There is no corresponding `diap`! But have `pure` : $A \Rightarrow P^A$

Applicative profunctors admit all previous constructions, and in addition:

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:

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

Categorical overview of standard functor classes

The "liftings" show the types of category's morphisms

class name	lifting's name and type signature	category's morphism
functor	fmap : $(A \Rightarrow B) \Rightarrow F^A \Rightarrow F^B$	$A \Rightarrow B$
filterable	fmapOpt : $(A \Rightarrow 1 + B) \Rightarrow F^A \Rightarrow F^B$	$A \Rightarrow 1 + B$
monad	flm : $(A \Rightarrow F^B) \Rightarrow F^A \Rightarrow F^B$	$A \Rightarrow F^B$
applicative	ap : $F^{A \Rightarrow B} \Rightarrow F^A \Rightarrow F^B$	$F^{A \Rightarrow B}$
contrafunctor	contrafmap : $(B \Rightarrow A) \Rightarrow F^A \Rightarrow F^B$	$B \Rightarrow A$
profunctor	dimap : $(A \Rightarrow B) \times (B \Rightarrow A) \Rightarrow F^A \Rightarrow F^B$	$(A \Rightarrow B) \times (B \Rightarrow A)$
contra-filterable	contrafmapOpt : $(B \Rightarrow 1 + A) \Rightarrow F^A \Rightarrow F^B$	$B \Rightarrow 1 + A$
Not yet considered:		
comonad	coflm : $(F^A \Rightarrow B) \Rightarrow F^A \Rightarrow F^B$	$F^A \Rightarrow B$

The laws are always just the category laws and the naturality laws
Need to define each category's composition and identity morphism

- Obtained a systematic picture of the "standard" type classes
- Some classes (e.g. contra-applicative) aren't covered by this scheme
- Some of the possibilities (e.g. "contramonad") don't actually work out

Exercises

1. Show that `pure` will be automatically a natural transformation when it is defined using `wu` as shown in the slides.

2. Use naturality of `pure` to show that $\text{pure}\, f \odot \text{pure}\, g = \text{pure}\,(f \circ g)$

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

4. Show that P^S is a monoid if S is a monoid and P is any applicative functor, contrafunctor, or profunctor.

5. Define an applicative instance for $F^A = 1 + \text{Int} \times A + A \times A \times A$.

6. Using applicative constructions, show without lengthy proofs that $F^A = G^A + H^{G^A}$ is applicative if G and H are applicative functors.

7. For any contrafunctor H^A, construction 5 says that $F^A \equiv H^A \Rightarrow A$ is applicative. Implement the code of `zip(fa, fb)` for this construction.

8. Show that the recursive functor $F^A \equiv 1 + G^{A \times F^A}$ is applicative if G^A is applicative and wu_F is defined recursively as $0 + \text{pure}_G\,(1 \times \text{wu}_F)$.