Chapter 8: Applicative and traversable functors

Part 2: Their laws and structure

Sergei Winitzki

Academy by the Bay

2018-06-07

Deriving the `ap` operation from `map2`

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

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

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

$$\begin{tikzcd}
& F^{B \Rightarrow Z} \arrow[dr, "\mathrm{ap}"] & \\
F^A \arrow[ur, "\mathrm{fmap}\, f"] \arrow[rr, "\mathrm{fmap}_2\, (f^{A \Rightarrow B \Rightarrow Z})"'] & & (F^B \Rightarrow F^Z)
\end{tikzcd}$$

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

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

$$\begin{tikzcd}
& F^{B \Rightarrow C \Rightarrow Z} \arrow[r, "\mathrm{ap}^{[B, C \Rightarrow Z]}"] & (F^B \Rightarrow F^{C \Rightarrow Z}) \arrow[dr, "\mathrm{fmap}_{F^B \Rightarrow ?} \mathrm{ap}^{[C,Z]}"] & \\
F^A \arrow[ur, "\mathrm{fmap}\, f"] \arrow[rrr, "\mathrm{fmap}_3\, (f^{A \Rightarrow B \Rightarrow C \Rightarrow Z})"'] & & & (F^B \Rightarrow F^C \Rightarrow F^Z)
\end{tikzcd}$$

- Using the infix syntax will get rid of $\mathrm{fmap}_{F^B \Rightarrow ?} \mathrm{ap}$ (see example code)

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 $\text{fmap2} : (A \times B \Rightarrow C) \Rightarrow F^A \times F^B \Rightarrow F^C$
- 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:

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

$$\begin{tikzcd}
 & F^{A \times B} \arrow[dr, "\text{fmap}\, f^{A \times B \Rightarrow C}"] & \\
F^A \times F^B \arrow[ur, "\text{zip}"] \arrow[rr, "\text{fmap2}\,(f^{A \times B \Rightarrow C})"'] & & F^C
\end{tikzcd}$$

- The functor F is "zippable" if such a `zip` exists

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: `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 $\text{pair} : (A \Rightarrow B \Rightarrow A \times B) = a^{A} \Rightarrow b^{B} \Rightarrow a \times b$
  - 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

$$\text{zip}\,(fa \times fb) = \text{app}\left((\text{pair}^{\uparrow} fa) \times fb\right)$$
$$\text{app}^{[B\Rightarrow C, B]} = \text{zip}^{[B\Rightarrow C, B]} \circ \text{fmap}\,(\text{eval})$$

$$\begin{tikzcd} & F^{(B\Rightarrow C)\times B} \arrow[dr, "\text{fmap(eval)}"] & \\ F^{B\Rightarrow C} \times F^{B} \arrow[ur, "\text{zip}"] \arrow[rr, "\text{app}^{[B\Rightarrow C,B]}"'] & & F^{C} \end{tikzcd}$$

- 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 $\text{fzip}\, p^{F^{A}} q^{F^{B}} = \text{ap}\left(\text{pair}^{\uparrow} p\right) q$, 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 for `map2`

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

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

```
map2 (
  cont1,
  cont2
) { (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 {
  x ← cont1.map(f)
  y ← cont2
} yield g(x, y)
```

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

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

```
for {
  y ← cont
} 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 `ap` and `zip`

- Set