Chapter 8: Applicative and traversable functors

Part 1: Practical examples

Sergei Winitzki

Academy by the Bay

2018-06-02

Motivation for applicative functors

Monads are inconvenient for expressing *independent* effects

- Effects will be performed *sequentially* even if they are independent:

```
x ← Future { c1 }
y ← Future { c2 }
z ← Future { c3 }
```

```
Future { c1 }.flatMap { x ⇒
  Future { c2 }.flatMap { y ⇒
    Future { c3 }.map { z ⇒ ...}
} }
```

We would like to parallelize independent computations automatically
We would like to accumulate *all* errors, rather than stop at the first one

- Changing the order of effects will (generally) change the result:

```
for {
  x ← List(1, 2)
  y ← List(10, 20)
} yield f(x, y)
// f(1, 10), f(1, 20), f(2, 10), f(2, 20)
```

```
for {
  y ← List(10, 20)
  x ← List(1, 2)
} yield f(x, y)
// f(1, 10), f(2, 10), f(1, 20), f(2, 20)
```

We would like to express a computation where effects are unordered

- This can be achieved if we have a method `map2` with type signature
  $\text{map2} : \left(F^A \times F^B\right) \Rightarrow (A \times B \Rightarrow C) \Rightarrow F^C$

Intuition: the `zip` operation on lists

- Simplify $\text{fmap2} : (A \times B \Rightarrow C) \Rightarrow F^A \times F^B \Rightarrow F^C$ by substituting $f = \text{id}^{A \times B \Rightarrow A \times B}$, 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

Deriving the `ap` operation

- 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 $\text{fmap}\,(\text{eval}) : F^{(B\Rightarrow C)\times B} \Rightarrow F^C$
- Define $\text{app}^{[B,C]} : F^{B\Rightarrow C} \times F^B \Rightarrow F^C \equiv \text{zip} \circ \text{fmap}\,(\text{eval})$
- The functions `zip` and `app` are computationally equivalent:
  - use $\text{pair} : (A \Rightarrow B \Rightarrow A \times B) = a^A \Rightarrow b^B \Rightarrow a \times b$
  - 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 $\text{fzip}\, p^{F^A} q^{F^B} = \text{ap}\,(\text{pair}^{\uparrow} p)\, q$, hence we can write as point-free: $\text{fzip} = \text{pair}^{\uparrow} \circ \text{ap}$. With explicit types: $\text{fzip}^{[A,B]} = \text{pair}^{\uparrow} \circ \text{ap}^{[B,A\Rightarrow B]}$.