Chapter 11: Computations in a functor context III. Monad transformers

Sergei Winitzki

Academy by the Bay

2019-01-05

Computations within a functor context: Combining monads

- Programs need to combine monadic effects
- "Effect" $\equiv$ what else happens in $A \Rightarrow M^B$ besides computing B from A
- Examples of effects for some standard monads:
  - `Option` – computation will have no result or a single result
  - `List` – computation will have zero, one, or multiple results
  - `Either` – computation may fail to obtain its result
  - `Reader` – computation needs to read an external context value
  - `Writer` – some value will be appended to a (monoidal) log
  - `Future` – computation will be scheduled to run later
- How to combine several effects in the same functor block (`for`/`yield`)?

```
(for { i ← 1 to n
       j ← 1 to n
       k ← 1 to n
  } yield f(i, j, k)
).sum
```

```
(1 to n).flatMap { i ⇒
  (1 to n).flatMap { j ⇒
    (1 to n).map { k ⇒
      f(i, j, k)
}}}.sum
```

- `map` replaces the last left arrow, `flatMap` replaces other left arrows

How `flatMap` works with lists

- consider `List(x1, x2, x3).flatMap(x ⇒ f(x))`
- assume that

```
f: X ⇒ List[Y]
f(x1) = List(y0, y1)
f(x2) = List(y2)
f(x3) = List(y3, y4, y5, y6)
```

- then the result is `List(y0, y1, y2, y3, y4, y5, y6)`

Exercises I

1. For a given `Set[Int]`, compute all subsets (w, x, y, z) of size 4 such that $w < x < y < z$ and $w + z = x + y$
2. Given 3 sequences xs, ys, zs of type `Seq[Int]`, compute all (x, y, z) such that $x \in xs$, $y \in ys$, $z \in zs$ and $x < y < z$ and $x + y + z < 10$
3. Solve the n-queens problem on an $3 \times 3 \times 3$ cube
4. Write a tiny library for arithmetic using `Future`'s; use it to compute $1 + 2 + ... + 100$ via `for`/`yield` and verify the result. E.g. implement:

```
const: Int ⇒ Future[Int]
add(x: Int): Int ⇒ Future[Int]
isEqual(x: Int): Int ⇒ Future[Boolean]
```

5. Read a file into a string and write it to another file using Java `Files` and `Paths` API. Use `Try` and `for`/`yield` to make this safe.
6. Given a semigroup W, make a semimonad out of $F^A \equiv E \Rightarrow A \times W$
7. Implement a semimonad instance for the (recursive) type constructor $F^A = A + A \times A + F^A + F^A \times F^A$
8. Find the largest prime number below 1000 via a simple Sieve of Eratosthenes; use the `State[S, Int]` monad with `S = Array[Boolean]`