Programowanie Funkcyjne

Rafał Włodarczyk

INA 4, 2025

Contents

1 Haskell 03.06

1.1 Monada Writer

$$\mathcal{M} = (M, \cdot, e) \tag{1}$$
$$W_{\mathcal{M}}(X) = X \times M \tag{2}$$
$$f : X \to Y \tag{3}$$
$$W_{\mathcal{M}}(f) : W_{\mathcal{M}}(X) \to W_{\mathcal{M}}(Y) \tag{4}$$
$$W_{\mathcal{M}}(f) : X \times M \to Y \times M \tag{5}$$
$$W_{\mathcal{M}}(f)(x, m) = (f(x), m) W_{\mathcal{M}}(Y^X) \quad \to W_{\mathcal{M}}(X) \to W_{\mathcal{M}}(Y) \tag{6}$$
$$(Y^X \times M) \to (X \times M) \to (Y \times M) \tag{7}$$
$$(f, m) < * > (x, n) = (f(x), m + n) \tag{8}$$
$$< | > \cdot W_{\mathcal{M}}(X) \to W_{\mathcal{M}}(Y) \to W_{\mathcal{M}}(X \times Y) \tag{9}$$
$$(x, m) < | > (y, n) = ((x, y), m + n) \tag{10}$$

```
(>>=) :: m a -> (a -> m b) -> mb
f :: X -> Y x M    (x,m) \in X x M
f (x) = (f1 (x), f2 (x))
f = (f1, f2)

((x,m) >>= f) = let (y, n) = f(x)
    in (y, m * n)

((x,m) >>= (f1, f2)) = (f1(x), m * f2(x))

pure x = return x = (x,e)
```

```
prue : a -> ma

Własności (aksjomaty) monadyczne w starych książkach.
(A1) (mx >>= return) = mx
(A2) ((return x) >>= f) = f x
(A3) ((mx >>= f) >>= g) = (mx >>= (\x -> f(x) >>= g))
(A4) mf <*> mx = do {f <- mf; x <- mx; return (fx)}
               = mf >>= (\f -> (mx >>= (\x -> return (fx))))
```

W parserach do gramatyk bezkontekstowych możemy wykorzystać jedynie <*>

```
Writer
return x = (x,e)
return = (id, const e)

(A1) : ((x,m) >>= return) =
       ((x,m) >>= (id, const e)) = (x, m * e) = (x, m)

(A2) : ((return x) >>= f) = (f1,f2) = (x,e) >>= (f1, f2) =
       (f1(x), e * f2(x)) = (f1(x),f2(x)) = f(x)

(A3) : (((x,m) >>= f) >> g) = ...
```

Pozostałe własności można sprawdzić automatycznie tą samą techniką.

1.2 Operator Kleisli'ego

Rybka. W języku Kleisli'ego własności monadyczne są jasno widoczne.

```
(>=>) :: (a -> mb) -> (b -> mc) -> (a -> mc)
(f >=> g) =def= (\x -> f(x) >>= g)
(f >=> return)(x) = (f(x) >>= return) =A1= f(x)
(A1') f >=> return = f

(return >=> g)(x) = (return x) >>= g =A2= g(x)
(A2') return >=> f = f
(A3') (f >=> g) >=> h = f >=> (g >=> h)

id >=> id // dla listy to konkatenacja listy list (operator spłaszczania)
```

1.3 Monada State

```
Fix S
ST (X) = (X x S)^S
f : X -> Y
ST (f) : ST(X) -> ST(Y)
       : (X x S)^S -> (Y x S)^S
ST(f) (phi) = (\s : S -> let (x,t) = phi(s) in
                         (f(x),t))
```

```
<*> = ??

(>>=) :: ma -> (a -> mb) -> mb
(>>=) : (X x S)^S -> (X -> (Y x S)^S) -> (Y x S)^S
(phi >>= f) = \s -> let(x,t) = phi(s),
                    (y,r) = f(x)(t) in
                    (y,r)
            = \s -> let(x,t) = phi(s)
                    (y,r) = u(f(x,t)) in
                    (y,r)
            = \s -> let(x,t) = phi(s) = u(f) . phi

u(f)(x,t) = (fx)(t)

(Zadanie) (f >=> g) = c (u(g) . u(f))

return x = \s -> (x,s)
u(return) = id
```