Definicje bezpunktowe

Inspiracja

Materiały do ćwiczeń będą oparte w dużej mierze na materiałach dr. Sławomira Bakalarskiego (do tego samego kursu).

Definicje bezpunktowe

Definicja bezpunktowa funkcji (*pointfree definition*) w Haskellu to definicja, która nie używa argumentu. Bardzo prostym przykładem jest przypisanie do funkcji `f` dodawania. Można to zrobić punktowo:

```
f x y = x + y
```

lub bezpunktowo:

```
f = (+)
```

Argument najbardziej z prawej

Jeśli chcemy doprowadzić definicję punktową do bezpunktowej, możemy to robić argument po argumencie, zaczynając od prawej (patrząc na argumenty po lewej stronie znaku = w definicji). Najprostszym przypadkiem jest sytuacja, kiedy argument najbardziej z prawej strony definicji występuje po prawej stronie znaku = również najbardziej z prawej:

```
doubleList list = map (*2) list
onlyOds list = filter odd list
```

Możemy wtedy po prostu nie zapisać tego argumentu (jeśli nie zmieni to "nawiasowania") i definicje będą równoważne:

```
doubleList = map (*2)
onlyOds = filter odd
```

Operator $

Operator $ służy do aplikacji funkcji, jednak ma **najniższy możliwy priorytet**. W praktyce oznacza to, że możemy zmniejszyć liczbę użytych nawiasów za pomocą $, na przykład

```
f x y z = sqrt (x+y+z)
```

jest równoważne z

```
f x y z = sqrt $ x+y+z
```

Powoduje to jednak, że nie możemy po prostu usunąć skrajnego prawego argumentu w **każdym** przypadku. W powyższym przykładnie definicja nie byłaby nawet poprawna. Samo to, że Haskell nie zgłosi nam błędu nie oznacza jednak, że definicja po usunięciu skrajnego argumentu jest równoważna. Przykładowo:

```
f x y =  (\a -> [a]) $ ((++) x) y
```

nie ma nawet takiego samego typu jak

```
f x   =  (\a -> [a]) $ ((++) x)
```

Operator złożenia

Przydatnym do przekształcania funkcji do postaci bezpunktowej będzie operator złożenia o sygnaturze

```
(.) :: (b->c)->(a->b)->a->c
```

Zapis `(f.g) x` jest równoważny `f(g x)`. Zamiast `(f.g) x` można też oczywiście użyć notacji prefiksowej: `((.) f g) x`.

Przykład

Zobaczmy, jak wygląda przekształcanie do postaci bezpunktowej na przykładzie, krok po kroku:

```
f x y = 3*x+y
f x y = (+) (3*x) y
f x = (+) (3*x)
f x = (+) ((*) 3 x)
f x = ((+).(*) 3) x
f = (+).(*) 3
```

Jeśli w definicji funkcji występuje lambda-wyrażenie, to powinniśmy je też sprowadzić do postaci bezpunktowej. Inaczej "doprowadzenie do postaci bezpunktowej" mogłoby wyglądać tak, że zamiast `f x y = <skomplikowanaDefinicja>` piszemy `f = (\x y -> <skomplikowanaDeficicja>)`.

flip

Kolejnym przydatnym narzędziem w doprowadzaniu funkcji do postaci bezpunktowej jest funkcja flip, która zamienia miejscami argumenty, tzn. `flip f x y` jest równoważne `f y x`. Przykładowo:

```
f x y = 3*y/x
f x y = (/) (3*y) x
f x y = flip (/) x (3*y)
f x = (flip (/) x).(3*)
f x = (.)(flip (/) x)(3*)
f x = flip (.) (3*) (flip (/) x)
f = (flip (.) (3*)).(flip (/))
```

curry i uncurry

Kolejnymi przydatnymi funkcjami są curry i uncurry. Mamy

```
curry :: ((a,b)->c)->a->b->c
(curry f) x y = f (x,y)
uncurry :: (a->b->c)-> ((a,b)->c)
(uncurry f)(x,y) = f x y
```

Przykładowo, zamiast

```
map (\(x,y) = x*y) [(1,2),(3,4),(5,6)]
```

Możemy użyć

```
map (uncurry (*)) [(1,2),(3,4),(5,6)]
```