Llenguatges de Programació

Interludi: aplicació vs composició

Gerard Escudero

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Contingut

- Aplicació vs composició
- Notació *point-free*
- Exemple amb 2 paràmetres

Aplicació vs composició

Aplicació:

```
($) :: (a -> b) -> a -> b

-- ($) f x

--    f    $    x
-- funció     valor
```

Composició:

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

-- (.) f g x

--    f    .     g
-- funció     funció
```

Relació entre l'aplicació i la composició:

```
\x -> f (g x)  ≡  \x -> f $ g x  ≡  f . g
```

Exemple:

```
appli2 f x = f $ f x

appli2 (*2) 2  👉  8
```

```
appli2 f = f . f

appli2 (*2) 2  👉  8
```

Contingut

- Aplicació vs composició
- Notació *point-free*
- Exemple amb 2 paràmetres

Notació *Point-free*

Notació per definir funcions sense explicitar els paràmetres que porta la funció. Normalment les definim utilitzant la composició de funcions.

Exemple:

Definició d'un funció que, donada una llista, ens torni el número de parell que conté.

```
numParells :: Integral a => [a] -> Int
numParells l = length (filter even l)
```

```
-- Point-free
numParells :: Integral a => [a] -> Int
numParells = length . filter even
```

Les dues definicions són completament equivalents:

```
numParells [1..5]  👉  2
```

Contingut

- Aplicació vs composició
- Notació *point-free*
- Exemple amb 2 paràmetres

Exemple amb 2 paràmetres

Definició d'un funció que, donada una llista i un enter, ens torni el número de vegades que apareix l'enter.

```
numVegades :: Eq a => a -> [a] -> Int
numVegades x l = length $ filter (== x) l

numVegades 3 [3,2,3]  👉  2
```

```
numVegades x l = (length . filter (== x)) l     -- canviem $ per .
```

```
numVegades x = length . filter (== x)           -- treiem l
```

```
numVegades x =  length . (filter ((==) x)))     -- treiem x de (== x)
```

```
numVegades x =  length . (filter $ (==) x))     -- canviem () per $
```

```
numVegades x =  length . (filter . (==)) x      -- caviem  $ per .
```

Exemple amb 2 paràmetres

En aquest punt tenim:

```
numVegades x =  length . (filter . (==)) x   -- necessitem quelcom com (f . g) x
```

Tenim els tipus simplificats i la `g`:

```
g :: Int -> [Int] -> [Int]
g = filter . (==)

numVegades :: Int -> [Int] -> Int
numVegades = f . g

(.) :: (b -> c) -> (a -> b) -> a -> c
a ≡ Int
b ≡ [Int] -> [Int]
c ≡ Int

length :: [Int] -> Int

f :: ([Int] -> [Int]) -> [Int] -> Int
f ??? length???
```

Exemple amb 2 paràmetres

Solució:

```
f :: ([Int] -> [Int]) -> [Int] -> Int
f = (length .)

f (map id) [2,4,1]  👉  3
```

```
numVegades x = ((length .) . (filter . (==))) x       -- apliquem f . g
```

```
numVegades = ((length .) . (filter . (==)))           -- treiem x
```

Test:

```
numVegades = (length .) .
             (filter . (==))

numVegades 3 [3,2,3]  👉  2
```

```
f = (length .)
g = filter . (==)

(f . g) 3 [3,2,3]  👉  2
```