

FUNKCIJSKO PROGRAMIRANJE



Funkcije višeg reda

Uvod

Za funkciju kažemo da je funkcija višeg reda ukoliko prima drugu funkciju kao argument ili vraća drugu funkciju kao rezultat.

```
dvaput :: (a → a) → a → a  
dvaput f x = f (f x)
```

dvaput je funkcija višeg reda jer uzima drugu funkciju kao argument

Zašto su korisne funkcije višeg reda?

- Zajedničke programerske paradigme mogu se izraziti kao funkcije unutar samog jezika.
- Domenski specifični jezici mogu se definirati kao skup funkcija višeg reda.
- Algebarska svojstva funkcija višeg reda mogu se koristiti za rezoniranje o programima.

***map* funkcija**

map je funkcija višeg reda koja primjenjuje funkciju na svaki element liste i vraća listu rezultata:

```
map :: (a → b) → [a] → [b]
```

Primjerice:

```
> map (+1) [1,3,5,7]  
[2,4,6,8]
```

map se može definirati pomoću komprehenzije listi:

```
map f xs = [f x | x ← xs]
```

Alternativno, u svrhu dokaza, *map* funkcija se također može definirati pomoću rekurzije:

```
map f []      = []  
map f (x:xs) = f x : map f xs
```

***filter* funkcija**

Filter je funkcija višeg reda koja vraća listu elementa koji zadovoljavaju predikat zadan funkcijom koju prosljeđujemo kao argument:

```
filter :: (a → Bool) → [a] → [a]
```

Primjerice:

```
> filter even [1..10]  
[2,4,6,8,10]
```

filter može biti definiran pomoću komprehenzije listi:

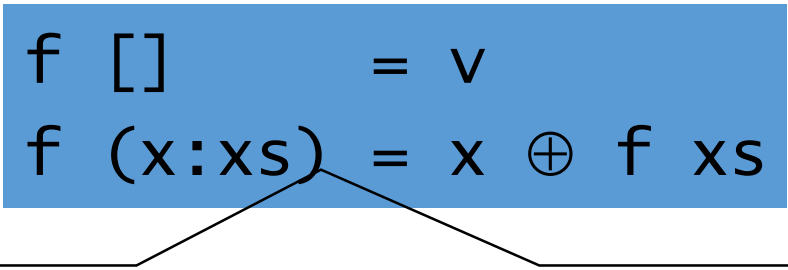
```
filter p xs = [x | x ← xs, p x]
```

Alternativno, može se definirati i pomoću rekurzije:

```
filter p [] = []  
filter p (x:xs)  
    | p x      = x : filter p xs  
    | otherwise = filter p xs
```

***foldr* funkcija**

Mnoge funkcije na listama mogu se definirati koristeći ovaj vrlo jednostavan obrazac rekurzije:



```
f []      = v
f (x:xs) = x ⊕ f xs
```

f pridružuje praznoj listi nekakvu vrijednost *v*,
a nepraznoj listi pridružuje \oplus primijenjen na
prvi element liste i *f* primijenjen na „rep liste”
(bez prvog elementa).

Primjerice:

```
sum [] = 0  
sum (x:xs) = x + sum xs
```

$v = 0$

$\oplus = +$

```
product [] = 1  
product (x:xs) = x * product xs
```

$v = 1$

$\oplus = *$

```
and [] = True  
and (x:xs) = x && and xs
```

$v = \text{True}$

$\oplus = \&\&$

Funkcija višeg reda foldr (eng. fold right) enkapsulira ovaj jednostavan obrazac rekurzije, primajući funkciju \oplus i vrijednost v kao argumente.

Primjerice:

```
sum = foldr (+) 0  
  
product = foldr (*) 1  
  
or = foldr (||) False  
  
and = foldr (&&) True
```

foldr funkcija također se može definirati koristeći rekurzije:

```
foldr :: (a → b → b) → b → [a] → b  
foldr f v []      = v  
foldr f v (x:xs) = f x (foldr f v xs)
```

Bolje je razmišljati o *foldr* nerekurzivno, zamjenjujući svaki $(:)$ s funkcijom, i $[]$ s danom vrijednošću.

Primjerice:

```
sum [1,2,3]  
=  
foldr (+) 0 [1,2,3]  
=  
foldr (+) 0 (1:(2:(3:[])))  
=  
1+(2+(3+0))  
=  
6
```

Zamjeni svaki (:) s (+) i [] s 0.

Primjerice:

```
product [1,2,3]  
=  
foldr (*) 1 [1,2,3]  
=  
foldr (*) 1 (1:(2:(3:[])))  
=  
1*(2*(3*1))  
=  
6
```

Zamjeni svaki (:) s (*) i [] s 1.

Drugi *foldr* primjeri

Iako *foldr* enkapsulira jednostavni princip rekurzije, može se koristiti za definiranje mnogo više funkcija nego što se to čini.

Prisjetimo se funkcije *length*:

```
length :: [a] → Int  
length []      = 0  
length (_:xs) = 1 + length xs
```

Primjerice:

`length [1,2,3]`
=
`length (1:(2:(3:[])))`
=
`1+(1+(1+0))`
=
`3`

Zamjeni svaki `(:)`
 $s \lambda_n \rightarrow 1+n$ i
`[]` s `0`.

Stoga imamo:

`length = foldr ($\lambda_n \rightarrow 1+n$) 0`

Prisjetimo se sada funkcije obrni:

```
obrni []      = []  
obrni (x:xs) = reverse xs ++ [x]
```

Primjerice:

```
obrni [1,2,3]  
=  
obrni (1:(2:(3:[])))  
=  
(([] ++ [3]) ++ [2]) ++ [1]  
=  
[3,2,1]
```

Zamjeni svaki $(:)$ s
 $\lambda x \text{ xs} \rightarrow \text{xs} ++ [x]$ i
 $[]$ s $[]$.

Dakle, imamo:

```
reverse = foldr ( $\lambda x \text{ } xs \rightarrow xs ++ [x]$ ) []
```

Na koncu, uočimo da funkcija $(++)$ ima elegantnu definiciju pomoću funkcije *foldr*:

```
(++ ys) = foldr (:) ys
```

Zamijeni
svaki $(:)$ s $(:)$
i $[]$ s ys .

Zašto je *foldr* koristan?

- Neke rekurzije na listama, kao što je suma, jednostavnije je definirati pomoću *foldr*.
- Svojstva funkcija koje su definirane pomoću *foldr* mogu se dokazati koristeći algebarska svojstva *foldr*.
- Napredne optimizacije programa mogu biti jednostavnije ako se koristi *foldr* umjesto eksplicitne rekurzije.

Ostale funkcije

Funkcija `(.)` (iz biblioteke) vraća kompoziciju dvije funkcije.

```
(.) :: (b → c) → (a → b) → (a → c)  
f . g = λx → f (g x)
```

Primjerice:

```
odd :: Int → Bool  
odd = not . even
```

Funkcije *all* (iz biblioteke) provjerava zadovoljava li svaki element liste dani predikat (zadan funkcijom koju prosljeđujemo kao argument).

```
all :: (a → Bool) → [a] → Bool  
all p xs = and [p x | x ← xs]
```

Primjerice:

```
> all even [2,4,6,8,10]  
True
```

Funkcija *any* (iz biblioteke) provjerava zadovoljava li barem jedan element u listi dani predikat (zadan funkcijom koju prosljeđujemo kao argument).

```
any :: (a → Bool) → [a] → Bool  
any p xs = or [p x | x ← xs]
```

Primjerice:

```
> any (== ' ') "abc def"  
True
```

Funkcija takeWhile (iz biblioteke) bira elemente iz liste sve dok je zadovoljen dani predikat (zadan funkcijom koju prosljeđujemo kao argument).

```
takeWhile :: (a → Bool) → [a] → [a]
takeWhile p [] = []
takeWhile p (x:xs)
    | p x          = x : takeWhile p xs
    | otherwise    = []
```

Primjerice:

```
> takeWhile (/= ' ') "abc def"
"abc"
```

Funkcija dropWhile (iz biblioteke) uklanja elemente iz liste sve dok je zadovoljen dani predikat (zadan funkcijom koju prosljeđujemo kao argument).

```
dropWhile :: (a → Bool) → [a] → [a]
dropWhile p [] = []
dropWhile p (x:xs)
  | p x      = dropWhile p xs
  | otherwise = x:xs
```

Primjerice:

```
> dropWhile (== ' ') " abc"
"abc"
```

Vježbe

- (1) Koji je još naziv za funkcije višeg reda koje vraćaju druge funkcije?
- (2) Koristeći `map` i `filter` zamijenite komprehenziju liste $[f\ x \mid x \leftarrow xs, p\ x]$ odgovarajućim izrazom.
- (3) Redefinirajte *map* *f* i *filter* *p* koristeći *foldr*.