Lambda-abstrakce, částečná aplikace, skládání funkcí, typování II

IB015 Neimperativní programování

Kolektiv cvičících IB015

Fakulta informatiky, Masarykova univerzita

podzim 2018

Opakování

Napište funkci `takeEvenElements :: [(Int, a)] -> [a]`,
která ze seznamu vybere pouze dvojice, jejichž první prvek je sudý,
a vrátí pouze druhý prvek.

Příklad:
`takeEvenElements [(1,'a'), (2,'b')]` $\rightsquigarrow$ `['b']`

Lambda-abstrakce

Anonymní funkce, umožňuje zapsat funkci přímo v místě volání.

- vhodné pro funkce jako `map`, `filter`
- `map (\x -> x ^ 2 + 2 * x + 1) [1, 2, 3]`
- obecně zapisujeme `\<parameters> -> function_body`
- parametrů může být více a oddělují se mezerami
- parametry můžou obsahovat vzory: `\(x, y) -> x + y`
- nelze zapsat ekvivalent víceřádkové definice nebo vyjádřit rekurzi

Lambda-abstrakce na příkladech

Zapište funkce s využitím lambda-abstrakce:
(Můžete si pomoci definicemi funkcí z minulého cvičení)

Příklad 2.2.19: Definujte funkci
`evens :: [Integer] -> [Integer]`, která ze seznamu vybere sudá čísla. Použijte funkci `filter`.

Příklad 2.2.20: S využitím funkce `map` a knihovní funkce `toUpper :: Char -> Char` z modulu `Data.Char` (tj. je třeba použít `import Data.Char`, na začátku souboru, nebo `:m + Data.Char` v interpretru) definujte novou funkci `toUpperStr`, která převádí řetězec písmen na řetězec velkých písmen, tj. `toUpperStr "bob"` $\rightsquigarrow^*$ `"BOB"`.

Částečná aplikace

Příklad 3.1.1: Co vyjadřuje výraz `min 6`? Napište ekvivalentní výraz pomocí `if`.

Částečná aplikace

Příklad 3.1.1: Co vyjadřuje výraz `min 6`? Napište ekvivalentní výraz pomocí `if`.

Poznámka: Nezapomeňte, že binární funkce i operátory mají prefixový i infixový tvar. Funkce převedeme z prefixového do infixového tvaru pomocí zpětných apostrofů. Naopak operátor převedeme do prefixového tvaru pomocí závorek.

Prefix	**Infix**
`mod 2 3`	`2 `mod` 3`
`min 3 4`	`3 `min` 4`
`(+) 4 2`	`4 + 2`

Částečná aplikace

Částečná aplikace

Umožňuje nám aplikovat funkci na méně parametrů, než očekává, a získat tak funkci, která bude očekávat zbylé parametry.

- `(+) 4`

Částečná aplikace

Částečná aplikace

Umožňuje nám aplikovat funkci na méně parametrů, než očekává, a získat tak funkci, která bude očekávat zbylé parametry.

- `(+) 4` (funkce, která očekává 1 parametr a přičte k němu 4)
- `(-) 3`

Částečná aplikace

Částečná aplikace

Umožňuje nám aplikovat funkci na méně parametrů, než očekává, a získat tak funkci, která bude očekávat zbylé parametry.

- `(+) 4` (funkce, která očekává 1 parametr a přičte k němu 4)
- `(-) 3` (funkce očekává 1 parametr a odečte ho od trojky)
- `max 2`

Částečná aplikace

Částečná aplikace

Umožňuje nám aplikovat funkci na méně parametrů, než očekává, a získat tak funkci, která bude očekávat zbylé parametry.

- `(+) 4` (funkce, která očekává 1 parametr a přičte k němu 4)
- `(-) 3` (funkce očekává 1 parametr a odečte ho od trojky)
- `max 2` (funkce očekává 1 parametr a vrátí větší z nich: 2 nebo zadaný parametr)

Částečná aplikace

Částečná aplikace

Umožňuje nám aplikovat funkci na méně parametrů, než očekává, a získat tak funkci, která bude očekávat zbylé parametry.

- `(+) 4` (funkce, která očekává 1 parametr a přičte k němu 4)
- `(-) 3` (funkce očekává 1 parametr a odečte ho od trojky)
- `max 2` (funkce očekává 1 parametr a vrátí větší z nich: 2 nebo zadaný parametr)

Operátorové sekce

Alternativní metoda zápisu částečné aplikace pro binární funkce.

- binární operátor a jeden argument v závorkách
- `(2 /)`

Částečná aplikace

Částečná aplikace

Umožňuje nám aplikovat funkci na méně parametrů, než očekává, a získat tak funkci, která bude očekávat zbylé parametry.

- `(+) 4` (funkce, která očekává 1 parametr a přičte k němu 4)
- `(-) 3` (funkce očekává 1 parametr a odečte ho od trojky)
- `max 2` (funkce očekává 1 parametr a vrátí větší z nich: 2 nebo zadaný parametr)

Operátorové sekce

Alternativní metoda zápisu částečné aplikace pro binární funkce.

- binární operátor a jeden argument v závorkách
- `(2 /)` (vydělí 2 svým argumentem – levá operátorová sekce)
- `(/ 2)`

Částečná aplikace

Částečná aplikace

Umožňuje nám aplikovat funkci na méně parametrů, než očekává, a získat tak funkci, která bude očekávat zbylé parametry.

- `(+) 4` (funkce, která očekává 1 parametr a přičte k němu 4)
- `(-) 3` (funkce očekává 1 parametr a odečte ho od trojky)
- `max 2` (funkce očekává 1 parametr a vrátí větší z nich: 2 nebo zadaný parametr)

Operátorové sekce

Alternativní metoda zápisu částečné aplikace pro binární funkce.

- binární operátor a jeden argument v závorkách
- `(2 /)` (vydělí 2 svým argumentem – levá operátorová sekce)
- `(/ 2)` (vydělí svůj argument 2 – pravá operátorová sekce)
- `(`mod` 2)`

Částečná aplikace

Částečná aplikace

Umožňuje nám aplikovat funkci na méně parametrů, než očekává, a získat tak funkci, která bude očekávat zbylé parametry.

- `(+) 4` (funkce, která očekává 1 parametr a přičte k němu 4)
- `(-) 3` (funkce očekává 1 parametr a odečte ho od trojky)
- `max 2` (funkce očekává 1 parametr a vrátí větší z nich: 2 nebo zadaný parametr)

Operátorové sekce

Alternativní metoda zápisu částečné aplikace pro binární funkce.

- binární operátor a jeden argument v závorkách
- `(2 /)` (vydělí 2 svým argumentem – levá operátorová sekce)
- `(/ 2)` (vydělí svůj argument 2 – pravá operátorová sekce)
- `(` `` `mod` `` `2)` (zjistí zbytek po dělení argumentu dvěma)
- pozor na `(- 1)`: speciální případ!

Skládání funkcí

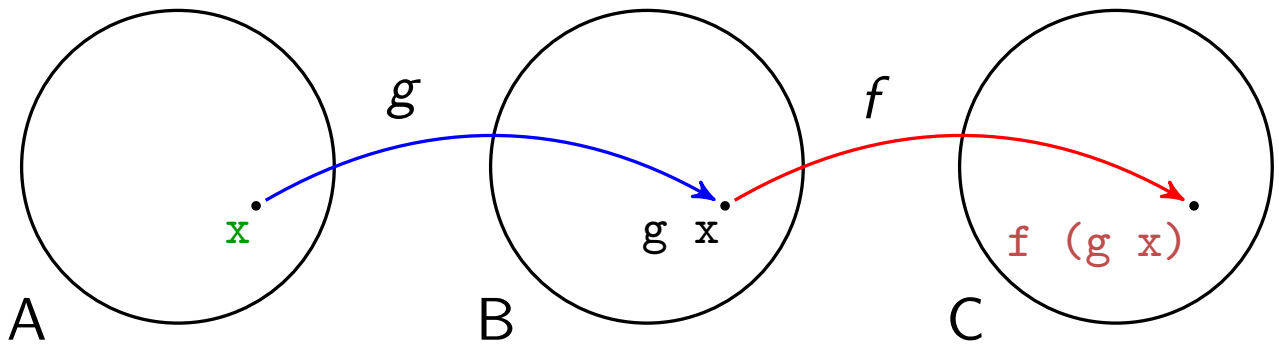
Jednou ze základních operací s funkcemi je jejich skládání.

- v matematice zapisujeme $f \circ g$, v Haskellu `f . g`,
  (`f` a `g` bereme jako unární funkce)
- aplikace složené funkce `(f . g) x` je ekvivalentní `f (g x)`
  - závorky jsou nutné, implicitní závorkování je `f . (g x)`
- operátor tečka má nejvyšší prioritu a je asociativní zprava
- `(.) ::`

Skládání funkcí

Jednou ze základních operací s funkcemi je jejich skládání.

- v matematice zapisujeme $f \circ g$, v Haskellu `f . g`,
  (`f` a `g` bereme jako unární funkce)
- aplikace složené funkce `(f . g) x` je ekvivalentní `f (g x)`
  - závorky jsou nutné, implicitní závorkování je `f . (g x)`
- operátor tečka má nejvyšší prioritu a je asociativní zprava
- `(.) :: (b -> c) -> (a -> b) -> (a -> c)`



Skládání funkcí v praxi

Příklad 3.2.1: Vyhodnoťte následující výrazy:

a) `((== 42) . (2 +)) 40`

b) `(even . (* 3) . max 4) 5`

c) `filter ((>= 2) . fst) [(1,"a"), (2,"b"), (3,"c")]`

Příklady

Napište funkce s využitím lambda-abstrakce, operátorových sekcí a skládání funkcí:

- `addOneOrNothing :: Integral a => [a] -> [a]`, která všechny sudé prvky ze seznamu zvětší o jedna. Liché nechá nezměněny.
- `justDivisibleBy3 :: Integral a => [a] -> [a]`, která ponechá v seznamu pouze prvky dělitelné číslem 3.
- `sumOfLists :: (Num a, Ord a) => [[a]] -> [a]`, která vrátí součty všech seznamů, které obsahují jen prvky větší než 5 (mohou se hodit funkce `all` a `sum`).

Typování funkcí: teorie

Intuitivní typování

- výraz posoudíme z hlediska toho, co dělá, a na základě toho určíme typ
- vhodné pro jednoduché výrazy

Algoritmické typování

- exaktní určení typu typovacím algoritmem
- zdlouhavější, ale funguje pro všechny výrazy

Typování funkcí I

Příklad 3.3.1: Určete typy výrazů:

a) `const True`

b) `const True False`

c) `(:[])`

d) `(:[]) True`

Zjednodušování před typováním

Výraz lze před otypováním vyhodnotit (zjednodušit). Ale pozor! Polymorfismus může při intuitivním vyhodnocení způsobit změnu typu:

- `head [id, not]`
- `min 2 (3.1 :: Float)`

Typování funkcí II

Příklad 3.3.4: Určete typy funkcí:

a) `swap (x,y) = (y,x)`

b) `caar = head . head`

c) `twice f = f . f`

Typování funkcí III

Příklad 3.3.6: Určete typy následujících funkcí:

a)

```
sayLength [] = "empty"
sayLength x  = "noempty"
```

b)

```
aOrX 'a' x = True
aOrX  _  x = x
```

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
  - Jaký typ má `head`?
  - Jaký typ má `map`?
- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
  - Jaký typ má `head`? `[a] -> a`
  - Jaký typ má `map`?
- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
  - Jaký typ má `head`? `[a] -> a`
  - Jaký typ má `map`? `(b -> c) -> [b] -> [c]`
- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
  - Jaký typ má `head`? `[a] -> a`
  - Jaký typ má `map`? `(b -> c) -> [b] -> [c]`

```
    map  ::  (b  →  c)  →  [b]  →  [c]
   head  ::  ([a]  →  a)
```

- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
  - Jaký typ má `head`? `[a] -> a`
  - Jaký typ má `map`? `(b -> c) -> [b] -> [c]`

  Rovnosti:

  ```
        map  ::  (b  →  c)  →  [b]  →  [c]
  head  ::  ([a]  →  a)
  ```

- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
  - Jaký typ má `head`? `[a] -> a`
  - Jaký typ má `map`? `(b -> c) -> [b] -> [c]`

  Rovnosti: `b = [a]`

```
map  ::  (b  →  c)  →  [b]  →  [c]

head ::  ([a] →  a)
```

- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
  - Jaký typ má `head`? `[a] -> a`
  - Jaký typ má `map`? `(b -> c) -> [b] -> [c]`

  Rovnosti: `b = [a]` , `c = a`

```
map  ::  (b  →  c)  →  [b]  →  [c]

head ::  ([a]  →  a)
```

- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
  - Jaký typ má `head`? `[a] -> a`
  - Jaký typ má `map`? `(b -> c) -> [b] -> [c]`

  Rovnosti: `b = [a]` , `c = a`

```
   map  ::  (b   → c)  →  [b]   →  [c]
   map  ::  ([a] → a)  →  [[a]] →  [a]
  head  ::  ([a] → a)
```

- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`?
  - Jaký typ má `head`? `[a] -> a`
  - Jaký typ má `map`? `(b -> c) -> [b] -> [c]`

  Rovnosti: `b = [a]` , `c = a`

`map`	`::`	`(b`	$\rightarrow$	`c)`	$\rightarrow$	`[b]`	$\rightarrow$	`[c]`
`map`	`::`	`([a]`	$\rightarrow$	`a)`	$\rightarrow$	`[[a]]`	$\rightarrow$	`[a]`
`head`	`::`	`([a]`	$\rightarrow$	`a)`				
`map head`	`::`					`[[a]]`	$\rightarrow$	`[a]`

- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`?
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`? `(b, c) -> c`
- Jaký typ má `(.)`?

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`? `(b, c) -> c`
- Jaký typ má `(.)`? `(e -> f) -> (d -> e) -> d -> f`

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`? `(b, c) -> c`
- Jaký typ má `(.)`? `(e -> f) -> (d -> e) -> d -> f`

```
     (.)  ::      (e→f)   →          (d→e)     →     d     →  f

map head  ::  ([[a]]→[a])
     snd  ::                    ((b,c)→c)
```

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`? `(b, c) -> c`
- Jaký typ má `(.)`? `(e -> f) -> (d -> e) -> d -> f`

Rovnosti:

```
     (.)  ::      (e→f)   →          (d→e)     →     d     →  f

map head  ::  ([[a]]→[a])
     snd  ::                    ((b,c)→c)
```

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`? `(b, c) -> c`
- Jaký typ má `(.)`? `(e -> f) -> (d -> e) -> d -> f`

Rovnosti: `e = [[a]], f = [a]`

```
     (.)  ::      (e→f)   →          (d→e)     →     d     →  f

map head  ::  ([[a]]→[a])
     snd  ::                    ((b,c)→c)
```

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`? `(b, c) -> c`
- Jaký typ má `(.)`? `(e -> f) -> (d -> e) -> d -> f`

Rovnosti: `e = [[a]]`, `f = [a]` , `d = (b,c)`, `e = c`

```
      (.)  ::      (e→f)   →          (d→e)     →     d     →  f

map head   ::  ([[a]]→[a])
     snd   ::                      ((b,c)→c)
```

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`? `(b, c) -> c`
- Jaký typ má `(.)`? `(e -> f) -> (d -> e) -> d -> f`

Rovnosti: `e = [[a]]`, `f = [a]` , `d = (b,c)`, `e = c`

```
     (.)  ::     (e→f)   →          (d→e)     →     d     →  f
     (.)  :: ([[a]]→[a]) → ((b, [[a]])→[[a]]) → (b,[[a]]) → [a]
map head  :: ([[a]]→[a])
     snd  ::                    ((b,c)→c)
```

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`? `(b, c) -> c`
- Jaký typ má `(.)`? `(e -> f) -> (d -> e) -> d -> f`

Rovnosti: `e = [[a]]`, `f = [a]` , `d = (b,c)`, `e = c`

```
     (.)  ::      (e→f)   →          (d→e)     →     d     →  f
     (.)  ::  ([[a]]→[a]) → ((b, [[a]])→[[a]]) → (b,[[a]]) → [a]
map head  ::  ([[a]]→[a])
     snd  ::                    ((b,c)→c)
     snd  ::                ((b, [[a]])→[[a]])
```

Algoritmické typování: příklad I

Jaký typ má výraz `map head . snd`?

- Jaký typ má `map head`? `[[a]] -> a`
- Jaký typ má `snd`? `(b, c) -> c`
- Jaký typ má `(.)`? `(e -> f) -> (d -> e) -> d -> f`

Rovnosti: `e = [[a]]`, `f = [a]` , `d = (b,c)`, `e = c`

```
     (.)  ::     (e→f)   →          (d→e)     →     d     →  f
     (.)  ::  ([[a]]→[a]) → ((b, [[a]])→[[a]]) → (b,[[a]]) → [a]
map head  ::  ([[a]]→[a])
     snd  ::                   ((b,c)→c)
     snd  ::                ((b, [[a]])→[[a]])
-------------------------------------------------------------------
  result  ::                                    (b,[[a]]) → [a]
```

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`**?**

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`?
  - Jaký typ má `maximum`?
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`?
    - Jaký typ má `4`?
    - Jaký typ má `(>)`?
  - Jaký typ má `maximum`?
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`?
    - Jaký typ má `4`? `(Num a) => a`
    - Jaký typ má `(>)`?
  - Jaký typ má `maximum`?
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`?
    - Jaký typ má `4`? `(Num a) => a`
    - Jaký typ má `(>)`? `(Ord b) => b -> b -> Bool`
  - Jaký typ má `maximum`?
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`?
    - Jaký typ má `4`? `(Num a) => a`
    - Jaký typ má `(>)`? `(Ord b) => b -> b -> Bool`

```
(>)     ::      (Ord b)       =>   b   ->   b   ->   Bool

4       ::      (Num a)       =>   a
```

  - Jaký typ má `maximum`?
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`?
    - Jaký typ má `4`? `(Num a) => a`
    - Jaký typ má `(>)`? `(Ord b) => b -> b -> Bool`

    Rovnosti: `b = a`

    ```
    (>)    ::      (Ord b)      =>   b   ->   b   ->   Bool

    4      ::      (Num a)      =>   a
    ```

  - Jaký typ má `maximum`?
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`?
    - Jaký typ má `4`? `(Num a) => a`
    - Jaký typ má `(>)`? `(Ord b) => b -> b -> Bool`

    Rovnosti: `b = a`

    ```
    (>)    ::    (Ord b)    =>   b   ->   b   ->   Bool
    (>)    ::    (Ord a)    =>   a   ->   a   ->   Bool
    4      ::    (Num a)    =>   a
    ```

  - Jaký typ má `maximum`?
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`?
    - Jaký typ má `4`? `(Num a) => a`
    - Jaký typ má `(>)`? `(Ord b) => b -> b -> Bool`

    Rovnosti: `b = a`

    ```
    (>)    ::        (Ord b)        =>   b   ->   b   ->   Bool
    (>)    ::        (Ord a)        =>   a   ->   a   ->   Bool
    4      ::        (Num a)        =>   a
    ------------------------------------------------------------
    (4>)   ::   (Ord a, Num a)      =>             a   ->   Bool
    ```

  - Jaký typ má `maximum`?
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`? `(Ord a, Num a) => a -> Bool`
  - Jaký typ má `maximum`?
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`? `(Ord a, Num a) => a -> Bool`
  - Jaký typ má `maximum`? `(Ord b) => [b] -> b`
  - Jaký typ má `(.)`?
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`? `(Ord a, Num a) => a -> Bool`
  - Jaký typ má `maximum`? `(Ord b) => [b] -> b`
  - Jaký typ má `(.)`? `(d -> e) -> (c -> d) -> c -> e`
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`? `(Ord a, Num a) => a -> Bool`
  - Jaký typ má `maximum`? `(Ord b) => [b] -> b`
  - Jaký typ má `(.)`? `(d -> e) -> (c -> d) -> c -> e`

```
(.) ::                    (d→e)    →   (c→d) → c  →  e

(4>) :: (Ord a, Num a) => (a→Bool)
maximum ::    (Ord b)    =>              ([b]→b)
```

- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`? `(Ord a, Num a) => a -> Bool`
  - Jaký typ má `maximum`? `(Ord b) => [b] -> b`
  - Jaký typ má `(.)`? `(d -> e) -> (c -> d) -> c -> e`

  Rovnosti: `d = a, e = Bool`

```
    (.) ::                  (d→e)    →   (c→d) → c  →  e

   (4>) :: (Ord a, Num a) => (a→Bool)
maximum ::    (Ord b)     =>            ([b]→b)
```

- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`? `(Ord a, Num a) => a -> Bool`
  - Jaký typ má `maximum`? `(Ord b) => [b] -> b`
  - Jaký typ má `(.)`? `(d -> e) -> (c -> d) -> c -> e`

  Rovnosti: `d = a, e = Bool` , `c = [b], d = b`

```
(.) ::                  (d→e)    →   (c→d) → c  →  e

(4>) :: (Ord a, Num a) => (a→Bool)
maximum ::    (Ord b)    =>            ([b]→b)
```

- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`? `(Ord a, Num a) => a -> Bool`
  - Jaký typ má `maximum`? `(Ord b) => [b] -> b`
  - Jaký typ má `(.)`? `(d -> e) -> (c -> d) -> c -> e`

  Rovnosti: `d = a, e = Bool , c = [b], d = b`

```
    (.) ::                    (d→e)    →   (c→d) → c  →  e
    (.) :: (Ord a, Num a) => (a→Bool) → ([a]→a) → [a] → Bool
   (4>) :: (Ord a, Num a) => (a→Bool)
maximum ::    (Ord b)     =>            ([b]→b)
```

- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`? `(Ord a, Num a) => a -> Bool`
  - Jaký typ má `maximum`? `(Ord b) => [b] -> b`
  - Jaký typ má `(.)`? `(d -> e) -> (c -> d) -> c -> e`

  Rovnosti: `d = a, e = Bool , c = [b], d = b`

```
    (.) ::                    (d→e)    →   (c→d) → c  →  e
    (.) :: (Ord a, Num a) => (a→Bool) → ([a]→a) → [a] → Bool
   (4>) :: (Ord a, Num a) => (a→Bool)
maximum ::    (Ord b)     =>            ([b]→b)
maximum :: (Ord a, Num a) =>            ([a]→a)
```

- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  - Jaký typ má `(4>)`? `(Ord a, Num a) => a -> Bool`
  - Jaký typ má `maximum`? `(Ord b) => [b] -> b`
  - Jaký typ má `(.)`? `(d -> e) -> (c -> d) -> c -> e`

  Rovnosti: `d = a`, `e = Bool`, `c = [b]`, `d = b`

```
    (.) ::                    (d→e)    →   (c→d) → c  →  e
    (.) :: (Ord a, Num a) => (a→Bool) → ([a]→a) → [a] → Bool
   (4>) :: (Ord a, Num a) => (a→Bool)
maximum ::    (Ord b)     =>            ([b]→b)
maximum :: (Ord a, Num a) =>            ([a]→a)
-------------------------------------------------------------
 result :: (Ord a, Num a) =>                      [a] → Bool
```

- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  `(Ord a, Num a) => [a] -> Bool`
- Jaký typ má `filter`?

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  `(Ord a, Num a) => [a] -> Bool`
- Jaký typ má `filter`? `(b -> Bool) -> [b] -> [b]`

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  `(Ord a, Num a) => [a] -> Bool`
- Jaký typ má `filter`? `(b -> Bool) -> [b] -> [b]`

```
filter    ::                        (b -> Bool) ->  [b]  ->  [b]

(4>).maximum :: (Ord a, Num a) => ([a] -> Bool)
```

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  `(Ord a, Num a) => [a] -> Bool`
- Jaký typ má `filter`? `(b -> Bool) -> [b] -> [b]`

Rovnosti: `b = [a]`

```
filter    ::                        (b -> Bool) ->  [b]  ->  [b]

(4>).maximum :: (Ord a, Num a) => ([a] -> Bool)
```

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  `(Ord a, Num a) => [a] -> Bool`
- Jaký typ má `filter`? `(b -> Bool) -> [b] -> [b]`

Rovnosti: `b = [a]`

```
   filter    ::                          (b -> Bool) ->  [b]  ->  [b]
   filter    :: (Ord a, Num a) => ([a] -> Bool) -> [[a]] -> [[a]]
(4>).maximum :: (Ord a, Num a) => ([a] -> Bool)
```

Algoritmické typování: příklad II

Příklad 3.3.13:
Jaký typ má výraz `filter ((4 >) . maximum)`?

- Jaký typ má `(4>) . maximum`?
  `(Ord a, Num a) => [a] -> Bool`
- Jaký typ má `filter`? `(b -> Bool) -> [b] -> [b]`

Rovnosti: `b = [a]`

```
   filter    ::                          (b -> Bool) ->  [b]  ->  [b]
   filter    :: (Ord a, Num a) => ([a] -> Bool) -> [[a]] -> [[a]]
(4>).maximum :: (Ord a, Num a) => ([a] -> Bool)
-------------------------------------------------------------------
   result    :: (Ord a, Num a) =>                  [[a]] -> [[a]]
```

Typování funkcí – lambda-abstrakce

Jaký typ má funkce: `\x y -> takeWhile y ('a': x)` ?

Typování funkcí – lambda-abstrakce

Jaký typ má funkce: `\x y -> takeWhile y ('a': x)` ?

- Vidíme, že funkce bere 2 argumenty (`x` a `y`) a vrací jeden.
  Takže zatím máme typ: `a -> b -> c`.

Typování funkcí – lambda-abstrakce

Jaký typ má funkce: `\x y -> takeWhile y ('a': x)` ?

- Vidíme, že funkce bere 2 argumenty (`x` a `y`) a vrací jeden. Takže zatím máme typ: `a -> b -> c`.
- V těle lambda-abstrakce je `x` použito ve výrazu `('a': x)`, z toho vyplývá, že `x` musí být typu `[Char]`. Dostáváme `[Char] -> b -> c`.

Typování funkcí – lambda-abstrakce

Jaký typ má funkce: `\x y -> takeWhile y ('a': x)` ?

- Vidíme, že funkce bere 2 argumenty (`x` a `y`) a vrací jeden. Takže zatím máme typ: `a -> b -> c`.
- V těle lambda-abstrakce je `x` použito ve výrazu `('a': x)`, z toho vyplývá, že `x` musí být typu `[Char]`. Dostáváme `[Char] -> b -> c`.
- Parametr `y` je použit jako první argument funkce `takeWhile :: (a -> Bool) -> [a] -> [a]`. Protože už víme, že `x` je typu `[Char]` a to je zároveň typ druhého argumentu `takeWhile`, vidíme, že typ `y` je `Char -> Bool`. Dostáváme `[Char] -> (Char -> Bool) -> c`.

Typování funkcí – lambda-abstrakce

Jaký typ má funkce: `\x y -> takeWhile y ('a': x)` ?

- Vidíme, že funkce bere 2 argumenty (`x` a `y`) a vrací jeden. Takže zatím máme typ: `a -> b -> c`.
- V těle lambda-abstrakce je `x` použito ve výrazu `('a': x)`, z toho vyplývá, že `x` musí být typu `[Char]`. Dostáváme `[Char] -> b -> c`.
- Parametr `y` je použit jako první argument funkce `takeWhile :: (a -> Bool) -> [a] -> [a]`. Protože už víme, že `x` je typu `[Char]` a to je zároveň typ druhého argumentu `takeWhile`, vidíme, že typ `y` je `Char -> Bool`. Dostáváme `[Char] -> (Char -> Bool) -> c`.
- Návratová hodnota funkce je stejná jako funkce `takeWhile`, čili `[a]`. Po naší specializaci je výsledkem `\x y -> takeWhile y ('a': x) :: [Char] -> (Char -> Bool) -> [Char]`