Zbierka príkladov na prevody medzi pointwise a poinfree tvarmi funkcií

Tomáš Szaniszlo

28.10.2010

1 Zadania príkladov

1. Preveďte `\x y -> f y x` na pointfree tvar.

   Riešenie:

   ```
   \x y -> f y x
   \x y -> flip f x y
   \x -> flip f x
   flip f
   ```

2. Preveďte `\x y -> f x (g y)` na pointfree tvar.

   Riešenie:

   ```
   \x y -> f x (g y)
   \x y -> (f x) (g y)
   \x y -> ((f x) . g) y
   \x -> (f x) . g
   \x -> (.g) (f x)
   \x -> ((.g) . f) x
   (.g) . f
   ```

3. Preveďte `(.) . (.)` na pointwise tvar, určte typy použitých parametrov a vhodne ich preznačte (funkcie na `f, g, ...`).

   Riešenie:

   ```
   (.) . (.)
   \x -> ((.) . (.)) x
   \x -> (.) ((.) x)
   \x y -> (.) ((.) x) y
   \x y -> ((.) x) . y
   \x y z -> (((.) x) . y) z
   \x y z -> ((.) x) (y z)
   \x y z -> (.) x (y z)
   ```

`\x y z -> x . (y z)`

`\x y z q -> (x . (y z)) q`

`\x y z q -> x ((y z) q)`

`\x y z q -> x (y z q)`

`\f g x y -> f (g x y)`

Typy parametrov, ktoré môžeme na základe tohto zápisu určiť, sú `f :: a -> b, g :: c -> d -> e, x :: c, y :: d`.

4. Dokážte, že `(.) . (.) . ... . (.)`, kde `(.)` sa vyskytuje n-krát ($n \geq 1$), má typ

   `(b -> c) -> (a1 -> a2 -> ... -> an -> b) -> a1 -> a2 -> ... -> an -> c`.

   Definujte funkciu `dots n`, ktorá bude vracať takúto funkciu pre každé $n \geq 1$. Ak to nie je možné, dokážte to.

   Riešenie: Tvrdenie budeme dokazovať pomocou matematickej indukcie podľa n. Pre $n = 1$ je `dots1 = (.)` a jej typ je `(a -> b) -> (c -> a) -> c -> b`. Po premenovaní typových premenných (`a, b, c` na `b, c, a1`) dostávame `(b -> c) -> (a1 -> b) -> a1 -> c`. Teda pre $n = 1$ tvrdenie platí. Teraz predpokladajme, že tvrdenie platí pre $n \geq 1$, t.j. `f = (.) . (.) . ... . (.)` s n `(.)` má typ

   `(b -> c) -> (a1 -> a2 -> ... -> an -> b) -> a1 -> a2 -> ... -> an -> c`.

   Potom chceme dokázať, že typ výrazu obsahujúceho o jednu `(.)` viac je

   `(b -> c) -> (a1 -> a2 -> ... -> a(n+1) -> b) -> a1 -> a2 -> ... -> a(n+1) -> c`.

   Funkcia, ktorej typ hľadáme, je vlastne `(.) . f`. Z infixovo zapísaného operátora skladania vieme, že typ tohto výrazu je `k -> m`, pričom typ prvého argumentu je `(.) :: l -> m` a druhého je `f :: k -> l`. Vieme, že `(.) :: (a' -> b') -> (c' -> a') -> c' -> b'`. Po porovnaní s `l -> m` dostávame

   `l = (a' -> b')` a

   `m = (c' -> a') -> c' -> b'`.

   Podobne porovnaním `k -> l` s typom `f` dostávame

   `k = (b -> c)` a

   `l = (a1 -> a2 -> ... -> an -> b) -> a1 -> a2 -> ... -> an -> c`.

   Porovnaním rovností pre `l` dostávame

   `a' -> b' = (a1 -> a2 -> ... -> an -> b) -> a1 -> a2 -> ... -> an -> c`.

   Odtiaľ

   `a' = a1 -> a2 -> ... -> an -> b` a

   `b' = a1 -> a2 -> ... -> an -> c`.

   Zistené rovnosti môžeme teraz využiť určení hľadaného typu:

`k -> m = (b -> c) -> (c' -> a') -> c' -> b' =`

`= (b -> c) -> (c' -> a1 -> a2 -> ... -> an -> b) -> c' -> a1 -> a2 -> ... -> an -> c.`

Po premenovaní typových premenných (`c', a1, ... , an` na `a1, a2, ... , an`) dostávame

`(b -> c) -> (a1 -> a2 -> ... -> a(n+1) -> b) -> a1 -> a2 -> ... -> a(n+1) -> c.`

A to je to, čo sme chceli dokázať.

Definovať funkciu `dots` nie je možné z nasledovného dôvodu. Jej typ, ako typ každéo výrazu, musí byť jednoznačne určený. Pozrime sa však, ako by vyzeralo `dots 1` a `dots 2`:

`dots 1 :: (b -> c) -> (a1 -> b) -> a1 -> c`

`dots 2 :: (b' -> c') -> (a1' -> a2' -> b') -> a1' -> a2' -> c'`

Musí platiť rovnosť zodpovedajúcich typov. Avšak máme `b -> c = b' -> c'`, čiže `b = b'`, ale zároveň `a1 -> b = a1' -> a2' -> b'`, odkiaľ `b = a2' -> b'`. Teda má platiť `b' = a2' -> b'`, čo je nezmysel, pretože podľa toho by bolo `b' = a2' -> a2' -> ... -> a2' -> b'` s ľubovoľným počtom `a2'`.

5. Dokážte alebo vyvráťte nasledujúce: `f (g x y)` ≡ `(f . g) x y`.

   Riešenie: Toto tvrdenie vyzerá byť analógiou k `\x -> f (g x)` ≡ `f . g`. Jeho správnosť podporuje napríklad aj správne vyhodnotenie výrazu `(id . div) 2 4` interpretom. Avšak podobne by sme mohli dať aj `(div . id) 2 4` a tu už zjavne `div (id 2 4)` nedáva zmysel. Ako by však vyzeralo správne odvodené `f (g x y)` v poinfree tvare? Nech `f, g` sú pevne dané. Potom môžeme voliť len `x, y`:

   ```
   \x y -> f (g x y)
   \x y -> f ((g x) y)
   \x -> f . (g x)
   \x -> (f.) (g x)
   \x -> ((f.) . g) x
   (f.) . g
   ```

   Teda správny tvar je `(f.) . g`.

6. Určte čo najpresnejší typ `f` vo výraze `(f.) . (.g)`.

   Riešenie: Sú dve možnosti, ako určíme typ výrazu. Môžeme ho upraviť na pointwise tvar, kde bude určenie už jednoduché alebo môžeme porovnávať typy jednotlivých funkcií. Ak by sme šli cestou upravovania, tak by to vyzeralo nasledovne:

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

`\x y -> f (x (g y))`.

Parametru `y` môžeme dať typ `a`. Odtiaľ máme `g :: a -> b, x :: b -> c` a konečne `f :: c -> d`. Teda po premenovaní máme `f :: a -> b`.

Otypovávanie by vyzeralo nasledovne:

$(.)_1$ `:: (a1 -> b1) -> (c1 -> a1) -> c1 -> b1`

$(.)_2$ `:: (a2 -> b2) -> (c2 -> a2) -> c2 -> b2`

$(.)_3$ `:: (a3 -> b3) -> (c3 -> a3) -> c3 -> b3`

To by boli jednotlivé `(.)` zľava. Dostali by sme `a2 -> b2 = ...` . TODO

7. Preveďte na pointwise tvar výraz `flip . map`.

   Riešenie:

   `flip . map`

   `\f -> flip (map f)`

   `\f x y -> flip (map f) x y`

   `\f x y -> map f y x`

   Tento výraz je síce syntakticky korektný, ale nastáva tu typový problém. V tom výraze sú implicitné zátvorky čiastočnej aplikácie nasledovne:

   `\f x y -> ((map f) y) x`.

   To v podstate znamená, že funkcia, v tomto prípade `map`, si zoberie koľko parametrov potrebuje, čiže dva parametre—funkciu `f :: a -> b` a zoznam `[a]`. Jej výsledkom bude zoznam typu [b], čiže dostaneme

   `\f x y -> ((map f y)::[b]) x`.

   Teraz však má dôjsť k aplikovaniu funkcie typu `[b]` na nejaký výraz. To samozrejme nie je možné, pretože výraz, ktorý je zoznam, nemôže byť funkcia. A keďže je výraz `x` dodaný, ale nemôže byť použitý, nastane chyba typovej kontroly.

   Tento príklad ukazuje, že nie každý výraz je korektne utvorený. Túto jeho vlastnosť zistíme tak, že sa pokúsime určiť jeho typ. Ak narazíme na problém, znamená to, že korektný nie je.

8. Určte, aké parametre musí mať funkcia `dist`, aby bol výsledný výraz ekvivalentný s výrazom `h (f x) (g x)`, kde `f, g, h, x` sú parametre.

   Riešenie: Mierne upravíme definíciu `dist`:

   `\f g x -> f x (g x)`

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

   Podobne upravíme cieľovú funkciu:

   `\f g h x -> h (f x) (g x)`

   `\f g h x -> (h (f x)) (g x)`

   `\f g h x -> (h (f x) . g) x`

   `\f g h x -> ((h . f) x . g) x`

   Po týchto (ekvivalentných) úpravách vidíme, že vieme zapísať hľadaný výraz pomocou `dist` nasledovne:

   `\f g h x -> dist (h . f) g x`