Prevody z pointfree tvaru na pointwise tvar

Tomáš Szaniszlo

2010-03-24 (v.2)

1 Príklad `(.(,)) . (.) . (,)`

Prevedenie z pointfree do pointwise tvaru výrazu `(.(,)) . (.) . (,)`.

```
(.(,)) . (.) . (,)
```

Teraz je funkcia v tvare `f . g . h`, kde `f = (.(,)), g = (.), h = (,)`. Použijeme definíciu skladania funkcie `(f . g . h) x ≡ f (g (h x))`, takže budeme musieť pridať na pravú stranu `x` (a takisto aj do zoznamu parametrov) a celú funkciu dať do zátvoriek (bez zátvoriek by mal prefix prednosť pred infixom a `f . g . h x` by malo implicitné zátvorky `f . (g . (h x))`, čo má ale iný význam). Dostaneme teda

```
\x -> ((.(,)) . (.) . (,)) x
```

```
\x -> (.(,)) ((.) ((,) x))
```

Teraz máme ako prvú funkciu pravú operátorovú sekciu operátora `(.)`, t.j. `(.(,))`. Platí `(.(,)) a ≡ a . (,)`, pričom v tomto prípade `a = ((.) ((,) x))`, preto dostávame

```
\x -> ((.) ((,) x)) . (,)
```

Zase sme sa dostali do tvaru `f . g`, kde `f = ((.) ((,) x)), g = (,)`, a preto použijeme `(f . g) x ≡ f (g x)` a postupujeme podobne ako pred chvíľou—celý výraz uzátvorkujeme, pridáme nový parameter y a prepíšeme podľa toho vzťahu pre skladanie dvoch funkcií:

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

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

Teraz máme vľavo `((.) ((,) x))`, čo je čiastočná aplikácia funkcie `(.)` na `((,) x)`. Tá však teraz už nie je potrebná a dokonca je aj menej výhodná, preto ju odstránime:

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

V tejto časti by sme už mohli prepísať zápisy tvaru `((,) a)` na `(a,)`. Či to spravíme teraz alebo neskôr, výsledok nezmení, hoci niektorá možnosť môže sprehľadniť ďalšie výrazy—podľa toho, komu čo viac vyhovuje.

V tomto výraze máme zase vľavo prefixový zápis skladania funckie. Ten prevedieme do infixu:

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

Zas využijeme `(f . g) x` ≡ `f (g x)`a veci s tým spojené:

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

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

Teraz sme už s úpravou výrazu v podstate hotoví a budú nasledovať len menšie prepisovania, konkrétne odstaňovania zbytočných čiastočných aplikácií a úpravy podľa `(,) a b` ≡ `(a, b)`:

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

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

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

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

A to je konečný (pointwise) tvar pôvodného výrazu, ktorého každá použitá funkcia už nevyžaduje žiadne ďalšie parametre.

2 Príklad `((,).) . (,)`

Prevedenie z pointfree do pointwise tvaru výrazu `((,).) . (,)`.

```
((,).) . (,)
```

Funkciu máme v tvare `f . g`. Obalíme ju zátvorkami a pridáme parameter x, aby sme mohli použiť prevod podľa `(f . g) x` ≡ `f (g x)`:

```
\x -> (((,).) . (,)) x
```

```
\x -> ((,).) ((,) x)
```

Teraz máme v ľavej funkcii `((,).)` operátorovú sekciu, ktorú rozpíšeme a dostaneme

```
\x -> (,) . ((,) x)
```

Zas máme funkciu v tvare, `f . g` a použijeme známu úpravu:

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

```
\x y -> (,) (((,) x) y)
```

Dostali sme funkciu `(,)`, ktorej dávame prvý parameter `(((,) x) y)`. V ňom máme nadbytočné zátvorky, ktoré odstránime, pričom dostaneme `((,) x y)` a následne `((x, y))` ≡ `(x, y)`, teda máme

```
\x y -> (,) (x, y)
```

Funkcia `(,)` vyžaduje ešte jeden parameter, preto jej ho dodáme a použitím `(,) a b = (a, b)` dostaneme konečný tvar:

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

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

3 Príklad `dist (curry id) id`

Prevedenie z pointfree do pointwise tvaru, kde `dist f g x = f x (g x)`.

```
dist (curry id) id
```

Keď sa pozrieme na typ funkcie `dist`, vidíme, že vyžaduje tri parametre (jej typ by bol `(a -> b -> c) -> (a -> b) -> a -> c`) a má zatiaľ len dva, preto dodáme jeden:

```
\x -> dist (curry id) id x
```

Ďalej použijeme definíciu funkcie `dist` a dostávame

```
\x -> (curry id) x (id x)
```

Tam môžeme hneď zjednodušiť `id x` na `x`:

```
\x -> (curry id) x x
```

Teraz musíme zistiť, čo robí funkcia `curry id`. Typ `curry` je `((a, b) -> c) -> a -> b -> c`. Keďže `id :: d -> d` je ako prvý parameter `curry`, musí platiť `(a, b) -> c = d -> d`, odkiaľ hneď vyplýva `(a, b) = c`, preto `curry id :: a -> b -> (a, b)`. Teda v tomto konkrétnom príklade má funkcia všetky parametre (dvakrát `x`). Vzhľadom na to, že `curry` podľa definície len mení parameter typu `(a, b)` na dva parametre `a, b` vidieť, že `(curry id) m n = (m, n) = (,) m n`. (Mimochodom presne táto funkcia sa preberala aj na cvičení.) Zostáva nám už len nahradiť túto funkciu a dostaneme výsledný pointwise tvar:

```
\x -> (x, x)
```

4 Príklad `uncurry (dist . ((.) (curry id)))`

Previesť z pointfree do pointwise tvaru, kde `dist` je ako v predchádzajúcom. Funkcia `uncurry :: (a -> b -> c) -> (a, b) -> c` má dva parametre, zatiaľčo v našom výraze má zatiaľ len jeden, preto jej pridáme aj parameter typu `(a, b)`:

```
\(x, y) -> uncurry (dist . ((.) (curry id))) (x, y)
```

Podľa definície `uncurry f (x, y) = f x y` dostaneme

```
\(x, y) -> (dist . ((.) (curry id))) x y
```

V tomto prípade máme niečo podobné ako `(f . g) x` $\equiv$ `f (g x)`, takže to vyzerá, že by mohlo platiť `(f . g) x y` $\equiv$ `f (g x y)`. Lenže to nie je vo všeobecnosti pravda. Totiž ak napríklad `f = (2*), g = (1+) :: Num a => a -> a`, v pravej strane to zjavne nebude sedieť, keďže `g` nemôže brať dva parametre. A zároveň to môže dopadnúť aj nasledovne: $\exists$`f, g: (f . g) x y` $\equiv$ `f (g x) y`. Príklady:

```
((1+) . mod) 4 3 ≡ (1+) (mod 4 3)

(mod . (1+)) 4 3 ≡ mod ((1+) 4) 3
```

Použijeme čiastočnú aplikáciu funkcie, t.j. pridáme zátvorky:

```
\(x, y) -> ((dist . ((.) (curry id))) x) y
```

Vnútri nich už môžeme použiť `(f . g) x` ≡ `f (g x)`:

```
\(x, y) -> ((dist . ((.) (curry id))) x) y
```

```
\(x, y) -> (dist (((.) (curry id)) x)) y
```

Vo výraze `((.) (curry id))` môžeme v danom kontexte vonkajšie zátvorky odstrániť, teda dostaneme

```
\(x, y) -> (dist ((.) (curry id) x)) y
```

Teraz sa zameriame na čierne zátvorky, t.j. na výraz `(.) (curry id) x`. Tu sa ukazuje, že `x` je druhý parameter operátora `(.)`, a teda je to v skutočnosti funkcia. (Čo je však v Haskelli stále výraz rovnocenný napr. s číslom.) Výraz môžeme prepísať do infixu podľa bodky: `curry id . x` (Nutnosť zátvoriek okolo `curry id` odpadá, pretože teraz je bodka infixová a prefix má prednosť pred infixom.) Zároveň z predchádzajúceho príkladu vieme, že `curry id` ≡ `(,)`, t.j. `(.) (curry id) x` ≡ `(,) . x`. Teda dostávame

```
\(x, y) -> (dist ((,) . x)) y
```

V tomto momente môžeme pôvodne pridané zelené zátvorky odstraniť a písať

```
\(x, y) -> dist ((,) . x) y
```

Vieme, že funkcia `dist` berie tri parametre, preto pridáme `z` a rozpíšeme ju podľa definície:

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

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

Použijeme podobný prístup ako predtým, t.j. vytvoríme čiastočnú aplikáciu funkcie `(,) . x` a použijeme `(f . g) x` ≡ `f (g x)`:

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

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

Tu môžeme posledne pridané zátvorky čiastočnej aplikácie odstrániť a po jednoduchej úprave dostávame konečný výsledok:

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

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

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

Vidíme, že `y` je takisto funkcia. Typ výrazu je `(a -> b, a -> c) -> a -> (b, c)`.

5 Niekoľko poznámok

Celkovo je dôležité dávať si dobrý pozor na správne prepisovanie výrazov, čiže neprihodiť/nevyhodiť nejakú zátvorku bodku alebo čokoľvek. Jedna z možností kontroly je dať si po jednotlivých úpravách otypovať novoodvodený výraz alebo doň dosadiť nejaké hodnoty (ak už vieme typ). Ak sa výsledok/typ nelíši od predchádzajúcich, je dobrá šanca, že krok bol správne vykonaný a je ekvivalentný (ale nemusí to tak byť!). Takisto výsledok daného výrazu môže napovedať o tom, čo tá funckia robí. Konkrétne u týchto príkladov, hlavne 1–3, to dobre vidieť, keďže okrem parametrov má výsledok pridané len dátové konštruktory. V prípade operátorov, ktoré sú komutatívne, nie je správne vyrábať ľubovoľnú sekciu, napr. `\x -> x + 3` $\rightsquigarrow$ `\x -> 3 + x` $\rightsquigarrow^*$ `(3+)`.

Prevod medzi jednotlivými tvarmi nie je jednoznačný, teda môže sa veľmi ľahko stať, že pri dvoch prevodoch vyjdú dva rôzne tvary výrazu, avšak oba budú správne. Napríklad `\x -> x` $\equiv$ `\x -> id x` $\equiv$ `id` $\equiv$ `id . id` $\equiv$ `flip const 3` $\equiv$ .... Samozrejme nie je nutné používať formálne parametre `x, y, ...`; je to len konvencia a môžu sa použiť aj iné (napríklad v príklade 4 by bolo vhodnejšie použiť `\(f, g) x -> (f x, g x)`).

5.1 Pointfree na pointwise

Pri počítaní takýchto príkladov sa treba držať niekoľkých zásad. Napríklad často sa dá využiť `(f . g) x` $\equiv$ `f (g x)`, prípadne obdobný vzťah pre viacero funkcií, čím sa zbavíme operácie skladania funkcií. Ďalej v prípade, že naľavo máme priamo funkciu, tak sa ju snažíme prepísať pomocou parametrov, ktoré za ňou nasledujú, podľa jej definície. Do každej funkcie musíme zadať toľko parametrov, koľko vyžaduje.

5.2 Pointwise na pointfree

Pri takýchto prevodoch väčšinou platí jedna hlavná zásada, ktorej podriaďujeme používané kroky. Totiž snažíme sa postupne presúvať parametre úplne doprava tak, aby za nimi nebolo nič (operátor, iný parameter, zátvorky, ...), a to robíme pre každý jeden v opačnom poradí, ako sú uvedené v zozname parametrov, napríklad vo výraze `\x y z -> x - z * (2 / y)` sa snažíme dostať napravo najprv `z`, a keď tam bude, tak ho môžeme "zmazať" zo zoznamu i z výrazu. V prípade, že premennú máme ako prvý parameter funkcie alebo operátora, výhodne sa dá použiť pravá operátorová sekcia alebo funkcia `flip`: `\x -> (f x . g 2)` $\rightsquigarrow$ `(. (g 2)) (f x)`, či `\x -> div x 2` $\rightsquigarrow^*$ `flip div 2`.