

Wstęp do Programowania potok funkcyjny

Marcin Kubica

2016/2017

Outline

- 1 Procedury wyższych rzędów jako abstrakcje konstrukcji programistycznych
 - Typy proceduralne i polimorfizm
 - Przykłady
 - Zastosowania procedur wyższych rzędów

Intuicje

- Procedury wyższych rzędów to procedury przetwarzające procedury.
- **Równouprawnienie procedur:**
z procedurami można robić wszystko to, co można robić z wartościami wszystkich pozostałych typów danych (z wyjątkiem porównywania).
- Procedury mogą być argumentami procedur lub ich wynikami.
- Jedno z zastosowań procedur wyższych rzędów: abstrakcja powtarzających się schematów postępowania.

Kilka prostych przykładów

Example

```
let p f = function x -> f x + f (2 * x);;  
val p : (int -> int) -> int -> int = <fun>
```

- Argumentem procedury p jest procedura f.
- Wynikiem jest λ -abstrakcja.
- Jak rozumieć typ procedury p?
- $(\text{int} \rightarrow \text{int}) \rightarrow (\text{int} \rightarrow \text{int})$.
Argument i wynik są typu $\text{int} \rightarrow \text{int}$.

Kilka prostych przykładów

Example

```
let twice f = fun x -> f (f x);;  
val twice : ('a -> 'a) -> ('a -> 'a) = <fun>
```

```
twice (fun x -> x * (x+1)) 2;;  
- : int = 42
```

```
twice (fun s -> "mocium panie, " ^ s) "me wezwanie";;  
- : string = "mocium panie, mocium panie, me wezwanie"
```

- f jest procedurą i można ją złożyć samą ze sobą.
- Argument i wynik f muszą być tego samego typu.
Może to być dowolny typ — $'a$.
- Procedura f jest typu $'a \rightarrow 'a$.
Procedura $twice$ jest typu $('a \rightarrow 'a) \rightarrow ('a \rightarrow 'a)$.

Polimorfizm

- **Polimorfizm** — ta sama procedura może być **wielu typów**.
- *Schemat typu* — zawiera zmienne typowe ('a, 'b, 'c, ...).
- Za zmienne typowe podstawiamy równocześnie dowolne typy (lub schematy).
- Ta sama zmienna — taki sam typ.
Różne zmienne — brak związku między typami.
- Ten sam mechanizm co w podstawianiu termów za zmienne, w termach ze zmiennymi.
- Uzgodnienie schematów typów = unifikacja termów.

Polimorfizm

Example

Podstawiając w typie $(\text{'a} \rightarrow \text{'a}) \rightarrow (\text{'a} \rightarrow \text{'a})$
za 'a typ int uzyskujemy $(\text{int} \rightarrow \text{int}) \rightarrow (\text{int} \rightarrow \text{int})$.

Podstawiając za 'a typ $\text{'a} \rightarrow \text{'a}$, uzyskujemy:
 $((\text{'a} \rightarrow \text{'a}) \rightarrow (\text{'a} \rightarrow \text{'a})) \rightarrow ((\text{'a} \rightarrow \text{'a}) \rightarrow (\text{'a} \rightarrow \text{'a}))$.

Polimorfizm

Example

```
let czterokrotnie f = twice twice f;;
val czterokrotnie : ('a -> 'a) -> ('a -> 'a) = <fun>
```

Jakiego typu jest tutaj procedura twice?

$$\begin{aligned} \text{czterokrotnie } f &= (\text{twice twice}) f = \\ &= \text{twice } (\text{twice } f) = \text{twice } f^2 = f^4 \end{aligned}$$

Drugie twice przetwarza f, a więc jest typu:

`('a -> 'a) -> ('a -> 'a)`.

Pierwsze twice przetwarza drugie twice, a więc jest typu:

`((('a -> 'a) -> ('a -> 'a)) -> (('a -> 'a) -> ('a -> 'a)))`.

Polimorfizm

Example

Składanie procedur:

```
let compose f g = function x -> f (g x);;  
val compose: ('a -> 'b) -> ('c -> 'a) -> ('c -> 'b) = <fun>
```

Trzy różne zmienne typowe.

Zależności między typami:

- wynik g i argument f muszą być tego samego typu,
- argument g i argument złożenia są tego samego typu,
- wynik f i wynik złożenia są tego samego typu.

Typ `compose` to najbardziej ogólny schemat typu zawierający te zależności.

Polimorfizm

Example

Wielokrotne składanie funkcji samej ze sobą:

```
let id x = x;;  
val id : 'a -> 'a = <fun>
```

```
let rec iterate n f =  
  if n = 0 then id else compose (iterate (n-1) f) f;;  
val iterate : int -> ('a -> 'a) -> ('a -> 'a) = <fun>
```

Alternatywna definicja twice:

```
let twice f = iterate 2 f;;  
val twice : ('a -> 'a) -> ('a -> 'a) = <fun>
```

Czy istnieją procedury wieloargumentowe?

- Typy proceduralne są postaci argument \rightarrow wynik.
- Typy procedur wieloargumentowych mają wiele strzałek:
$$\tau_1 \rightarrow \tau_2 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau_{n+1} =$$
$$= \tau_1 \rightarrow (\tau_2 \rightarrow \dots \rightarrow (\tau_n \rightarrow \tau_{n+1}) \dots).$$
- Można je interpretować na dwa sposoby:
 - procedura n -argumentowa,
 - procedura jednoargumentowa, której wynikiem jest procedura typu: $\tau_2 \rightarrow \dots \rightarrow \tau_{n+1}$.
- Przy odpowiedniej kolejności argumentów możemy podać tylko część z nich, otrzymując w wyniku przydatną procedurę.

Czy istnieją procedury wieloargumentowe?

Example

- Jeden argument, który jest parą liczb:

```
let plus (x, y) = x + y;;  
val plus : (int * int) -> int = <fun>
```

- Dwa argumenty, które są liczbami:

```
let plus x y = x + y;;  
val plus : int -> (int -> int) = <fun>
```

- Można ten typ interpretować jako `int -> (int -> int)`.
Procedura jednoargumentowa, której wynikiem jest procedura jednoargumentowa.

```
let plus = function x -> function y -> x + y;;  
val plus : int -> (int -> int) = <fun>
```

Czy istnieją procedury wieloargumentowe?

Example

Podanie części argumentów może mieć sens:

```
let plus x y = x + y;;  
val plus : int -> (int -> int) = <fun>
```

```
let inc = plus 1;;  
val inc : int -> int = <fun>
```

Czy istnieją procedury wieloargumentowe?

Example

Obie formy przekazywania argumentów są równoważne.
Dowodem procedury przekształcające jedną postać w drugą i odwrotnie.

```
let curry f = function x -> function y -> f (x, y);;  
val curry : (('a * 'b) -> 'c) -> ('a -> 'b -> 'c) = <fun>
```

```
let uncurry f = function (x, y) -> f x y;;  
val uncurry : ('a -> 'b -> 'c) -> (('a * 'b) -> 'c) = <fun>
```

Example

Przedstawione wcześniej procedury można zdefiniować i tak:

```
let twice f x = f (f x);;  
val twice : ('a -> 'a) -> ('a -> 'a) = <fun>
```

```
let compose f g x = f (g x);;  
val compose : ('a -> 'b) -> ('c -> 'a) -> ('c -> 'b) = <fun>
```

```
let curry f x y = f (x, y);;  
val curry : (('a * 'b) -> 'c) -> ('a -> 'b -> 'c) = <fun>
```

```
let uncurry f (x, y) = f x y;;  
val uncurry : ('a -> 'b -> 'c) -> (('a * 'b) -> 'c) = <fun>
```

Sumy częściowe szeregów

Example

- Przybliżanie szeregów przez ich sumy częściowe.
- Zdefiniujmy procedurę obliczającą sumę częściową dowolnego szeregu.
- Elementy szeregu są zadane jako procedura.

```
let szereg f n =  
  let rec sum a n =  
    if n = 0 then a else sum (a +. f n) (n - 1)  
  in  
    sum 0.0 n;;  
val szereg : (int -> float) -> (int -> float) = <fun>
```


Sumy częściowe szeregów

Example

Przybliżamy szereg: $\sum_{i=1}^{\infty} \frac{1}{(4i-3)(4i-1)} = \frac{\pi}{8}$.

```
let szereg_pi_8 n =  
  szereg  
    (function i ->  
      1. /. ((4. *. float i -. 3.) *. (4. *. float i -. 1.)))  
    n;;  
val szereg_pi_8 : int -> float = <fun>  
  
let pi = 8. *. szereg_pi_8 1000;;  
val pi : float = 3.14109265362103818
```

Abstrakcja proceduralna

- Procedury — abstrakcja powtarzających się fragmentów kodu.
- Procedury wyższych rzędów — abstrakcja powtarzających się schematów, które różnią się fragmentami kodu wewnątrz nich.

Szukanie zer funkcji

Example

- Dwie metody szukania zer funkcji:
 - przez bisekcję i
 - metoda Newtona (stycznych).
- Metoda przez bisekcję — podział przedziału, na końcach którego funkcja przyjmuje przeciwne znaki.
- Metoda stycznych — iteracja następującego procesu:
 - w punkcie będącym aktualnym przybliżeniem zera funkcji rysujemy styczną do wykresu funkcji,
 - kolejnym przybliżeniem jest punkt przecięcia stycznej z osią X .
- Obie metody polegają na polepszaniu pewnego przybliżenia, aż do uzyskania satysfakcjonującego wyniku.

Schemat iteracyjnego przybliżania

Example

Ogólny schemat iteracji przybliżającej wynik:

```
let rec iteruj poczatek popraw czy_dobre wynik =  
  if czy_dobre poczatek then  
    wynik poczatek  
  else  
    iteruj (popraw poczatek) popraw czy_dobre wynik;;  
val iteruj : 'a -> ('a -> 'a) -> ('a -> bool) ->  
                                     ('a -> 'b) -> 'b = <fun>
```

Example

Metoda przez bisekcję i Newtona jako instance iteracji:

Parametr	Metoda Newtona	Metoda przez bisekcję
poczatek	punkt w pobliżu zera	końce przedziału zawierającego zero
popraw	przecięcie stycznej do wykresu funkcji z osią X	podział przedziału na pół
czy_dobre	wartość funkcji w punkcie jest bliska zera	wartość funkcji na końcu przedziału jest bliska zera
wynik	dany punkt	koniec przedziału

Metoda przez bisekcję

Example

- Dana funkcja f oraz dwa punkty, l i p , w których f przyjmuje wartości przeciwnych znaków.
- Gdzieś pomiędzy l i p funkcja f ma zero.
- **Uproszczenie:** przyjmujemy, że $f(l) \leq 0 < f(p)$, dopuszczamy aby $l < p$ lub $l > p$.

Metoda przez bisekcję

Example

Procedura bisekcja bada wartość f w punktach l i p ,
i wywołuje szukaj zgodnie z przyjętym uproszczeniem:

```
let bisekcja f l p =  
  let rec szukaj l p = ...  
  in  
    let wartosc_l = f l  
    and wartosc_p = f p  
    in  
      if ujemne wartosc_l && dodatnie wartosc_p then  
        szukaj l p  
      else  
        szukaj p l;;  
val bisekcja : (float -> float) -> float -> float ->  
float = <fun>
```

Metoda przez bisekcję

Example

Procedura szukaj korzysta z procedury iteruj:

```
let szukaj l p =  
  let czy_dobre x = ...  
  and popraw x = ...  
  in  
    iteruj  
      (l, p)  
      popraw  
      czy_dobre  
      fst
```


Metoda przez bisekcję

Example

- x jest dobrym przybliżeniem, gdy $|f\ x| < \varepsilon$.
- Bisekcja — badamy znak f po środku między l i p .

```
let czy_dobre x =  
  abs_float (f (fst x)) < epsilon  
and popraw x =  
  let l = fst x  
  and p = snd x  
  in  
    let srodek = average l p  
    in  
      if dodatnie (f srodek) then  
        (l, srodek)  
      else  
        (srodek, p)
```

Metoda przez bisekcję

Example

Przykład: zastosowanie metody przez bisekcję do pierwiastkowania:

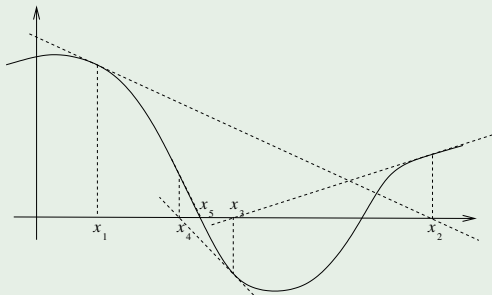
```
let sqrt a =  
  let f x = a -. square x  
  in bisekcja f 0.0 (a +. 1.0);;  
val sqrt : float -> float = <fun>  
  
sqrt 42.0;;  
- : float = 6.48074069840786
```

Metoda Newtona

Example

Uproszczenia:

- Zakładamy, że dana funkcja jest różniczkowalna.
- Pomijamy kwestię własności stopu i dzielenia przez 0.
- Styczna do wykresu funkcji w x przecina oś X w punkcie $x - \frac{(f \ x)}{(f' \ x)}$.



Metoda Newtona

Example

Metoda Newtona jako iteracyjne przybliżenie:

```
let newton f x =  
  let p = pochodna f  
  in  
    let czy_dobre x = abs_float (f x) < epsilon  
    and popraw x = x -. (f x) /. (p x)  
    in  
      iteruj x popraw czy_dobre id;;  
val newton : (float -> float) -> float -> float = <fun>
```

Metoda Newtona

Example

- *Metoda Newtona* oznacza też metodę liczenia pierwiastków kwadratowych.
- Jest to szczególny przypadek metody Newtona znajdowania zer, dla funkcji $f(x) = x^2 - a$.
Funkcja ta ma zera w punktach $\pm\sqrt{a}$.

- $f'(x) = 2x$, czyli kolejne przybliżenie to:

$$x - \frac{f(x)}{f'(x)} = x - \frac{x^2 - a}{2x} = \frac{2x^2 - x^2 + a}{2x} = \frac{x^2 + a}{2x} = \frac{x + \frac{a}{x}}{2}$$

- Otrzymujemy dokładnie metodę pierwiastkowania Newtona.
- Przybliżenie zaczynamy w dowolnym dodatnim punkcie, np. 1.

Metoda Newtona

Example

```
let sqrt a =  
  let f x = a -. square x  
  in newton f 1.;;  
val sqrt : float -> float = <fun>  
  
sqrt 17.64;;  
- : float = 4.200000000000023643
```

Punkty stałe funkcji

Example

- Punkt stały funkcji f to taki x , że $f(x) = x$.
- Dla niektórych funkcji f i x -ów, ciąg $x, f(x), f^2(x), f^3(x), \dots$ jest zbieżny do pewnego punktu stałego f .
- Jest tak np. dla przekształceń zwężających.
- **Fakt:** Jeżeli f jest funkcją ciągłą, oraz ciąg $x, f(x), f^2(x), \dots$ jest zbieżny, to $\lim_{i \rightarrow \infty} f^i(x)$ jest punktem stałym f .

Punkty stałe funkcji

Example

Przybliżanie punktów stałych za pomocą procedury iteruj:

```
let punkt_staly f x =  
  let blisko x = abs_float (x -. f x) < epsilon  
  in  
    iteruj x f blisko f;;  
val punkt_staly: (float->float) -> float->float = <fun>  
  
punkt_staly cos 1.0;;  
0.73908...
```


Punkty stałe funkcji

Example

- Pierwiastkowanie za pomocą obliczania punktów stałych.

Punkt stały funkcji $y \rightarrow \frac{x}{y}$ to $\pm\sqrt{x}$.

- Obliczenie:

```
punkt_staly (function y -> x /. y) 1.0;;
```

zapętla się:

$$1 \rightarrow \frac{x}{1} = x \rightarrow \frac{x}{x} = 1 \rightarrow \dots$$

Punkty stałe funkcji

Example

Wyłumienie przez uśrednienie:

- Uśrednienie funkcji f , to funkcja $x \mapsto \frac{f(x)+x}{2}$.
- Uśrednienie f ma dokładnie takie same punkty stałe co f .
- Szukamy punktów stałych uśrednienia f , zamiast f .

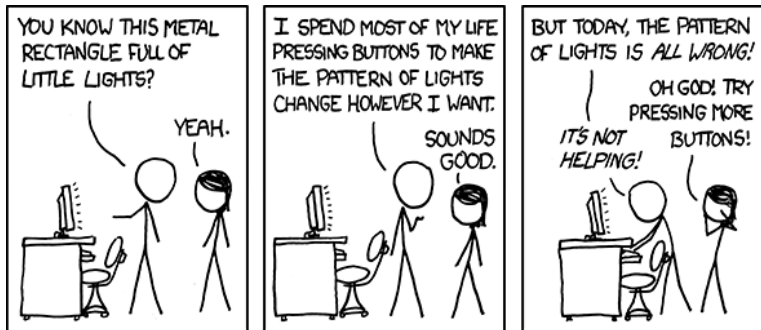
```
let usrednienie f =  
  function x -> average x (f x);;  
val usrednienie : (float -> float) -> float -> float = <fun>
```

```
let sqrt x =  
  punkt_staly (usrednienie (function y -> x/.y)) 1.0;;  
val sqrt : float -> float = <fun>
```

- Otrzymaliśmy ... metodę pierwiastkowania Newtona.

- Naturalny sposób implementacji pojęć matematycznych (np. składanie funkcji i różniczkowanie funkcji).
- Jeśli dane są procedurami, to operacje na nich są procedurami wyższych rzędów. Na przykład, predykaty i operacje logiczne.
- Abstrakcja powtarzających się schematów procedur (np. iteruj).
- Standardowo zdefiniowane procedury wyższych rzędów przetwarzające listy.
(Zobaczmy to w kolejnym wykładzie.)

Deser



<http://xkcd.com/722/>