

Wstęp do Programowania potok funkcyjny

Marcin Kubica

2010/2011

Outline

- 1 Funktory
 - Pojęcie funktora
 - Przykłady funktorów
 - Funktory wyższych rzędów

Intuicje

Intuicje

- Funktory to moduły sparametryzowane (innymi modułami).
- Wykorzystywanie jednych modułów przez inne:
 - tradycyjne: moduł korzysta z konkretnych innych modułów,
 - funktory: moduł korzysta z modułów zapewniających określone interfejsy, moduły o tych interfejsach stają się parametrami funktora.
- Interfejs = specyfikacja
Information hiding — okeślamy co mają dostarczać wykorzystywane moduły, a nie które konkretnie to są moduły.
- *Separation of concerns*: specyfikacja, implementacja, sposób łączenia modułów.
- Funktory = funkcje na strukturach.
- Szablony (ang. *templates*) i dużo więcej ...

Składnia funktorów

Definition

```

⟨sygnatura⟩ ::= functor ( ⟨Identyfikator⟩ : ⟨sygnatura⟩ )
                                     -> ⟨sygnatura⟩ |
...
⟨struktura⟩ ::= functor ( ⟨Identyfikator⟩ : ⟨sygnatura⟩ )
                                     -> ⟨struktura⟩ |
⟨struktura⟩ ( ⟨struktura⟩ ) |
...
⟨definicja⟩ ::= module ⟨Identyfikator⟩
                ( ⟨Identyfikator⟩ : ⟨sygnatura⟩ )
                [ : ⟨sygnatura⟩ ] ≡ ⟨struktura⟩ |
...

```

Pojęcie przestrzeni metrycznej

Definition

Przestrzeń metryczna, to zbiór punktów X , wraz z *metryką* $d : X \times X \rightarrow [0; \infty)$ określającą odległości między punktami.

Metryka musi spełniać następujące warunki:

- $d(a, b) = 0 \Leftrightarrow a = b$,
- $d(a, b) = d(b, a)$,
- $d(a, b) + d(b, c) \geq d(a, c)$,

dla dowolnych $a, b, c \in X$.

Przestrzenie metryczne

Example

Przykłady metryk na płaszczyźnie \mathbb{R}^2 :

- metryka euklidesowa:

$$d_2((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- metryka miejska (Manhattan):

$$d_1((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$$

- metryka maksimum:

$$d_\infty((x_1, y_1), (x_2, y_2)) = \max(|x_1 - x_2|, |y_1 - y_2|)$$

Zdziwienia dnia

Zdziwienia dnia

- Jaki kształt mają okręgi w tych metrykach?
- Na płaszczyźnie metryki d_1 i d_∞ tworzą przestrzenie izomorficzne. Podaj przekształcenie układu współrzędnych, które przekształca jedną przestrzeń w drugą.
- Czy przestrzenie metryczne z metrykami d_1 i d_∞ też są izomorficzne? Jaki kształt mają sfery w tych przestrzeniach?

Przestrzenie metryczne

Example

Zaimplementujmy pojęcie przestrzeni metrycznej oraz opisane trzy przestrzenie metryczne.

```
module type METRIC_SPACE =  
  sig  
    type t  
    val d: t -> t -> float  
  end;;  
  
module Euclidian_Plane : METRIC_SPACE =  
  struct  
    type t = float * float  
    let square x = x *. x  
    let d (x1, y1) (x2, y2) =  
      sqrt (square (x1 -. x2) +. square (y1 -. y2))  
  end;;
```


Przestrzenie metryczne

Example

```
module Manhattan_Plane : METRIC_SPACE =
  struct
    type t = float * float
    let d (x1, y1) (x2, y2) =
      abs_float (x1 -. x2) +. abs_float (y1 -. y2)
    end;;

module Max_Plane : METRIC_SPACE =
  struct
    type t = float * float
    let d (x1, y1) (x2, y2) =
      max (abs_float (x1 -. x2)) (abs_float (y1 -. y2))
    end;;
```

Produkt przestrzeni metrycznych

Example

- Funktor produktu kartezjańskiego przestrzeni metrycznych.
- Różne sposoby składania metryk:

```
module Manhattan_Product (S1 : METRIC_SPACE)
                        (S2 : METRIC_SPACE) : METRIC_SPACE =
  struct
    type t = S1.t * S2.t
    let d (x1, y1) (x2, y2) =
      S1.d x1 x2 +. S2.d y1 y2
  end;;
```

Produkt przestrzeni metrycznych

Example

```
module Euclidian_Product (S1 : METRIC_SPACE)
    (S2 : METRIC_SPACE) : METRIC_SPACE =
  struct
    type t = S1.t * S2.t
    let square x = x *. x
    let d (x1, y1) (x2, y2) =
      sqrt (square (S1.d x1 x2) +. square (S2.d y1 y2))
  end;;

module Max_Product (S1 : METRIC_SPACE)
    (S2 : METRIC_SPACE) : METRIC_SPACE =
  struct
    type t = S1.t * S2.t
    let d (x1, y1) (x2, y2) =
      max (S1.d x1 x2) (S2.d y1 y2)
  end;;
```

Produkt przestrzeni metrycznych

Example

\mathbb{R} z metryką $|\cdot|$:

```
module Line : METRIC_SPACE =  
  struct  
    type t = float  
    let d x y = abs_float (x -. y)  
  end;;
```

Produkt przestrzeni metrycznych

Example

Alternatywna definicja trzech przestrzeni metrycznych na płaszczyźnie:

```
module Manhattan_Plane =  
  Manhattan_Product (Line) (Line);;  
  
module Euclidian_Plane =  
  Euclidian_Product (Line) (Line);;  
  
module Max_Plane =  
  Max_Product (Line) (Line);;
```

Kolejka priorytetowa

Example

- Kolejka priorytetowa może być sparametryzowana porządkiem liniowym określonym na elementach.
- Porządek taki ma sygnaturę:

```
type porownanie = Mniejsze | Rowne | Wieksze;;  
  
module type PORZADEK_LINIOWY =  
  sig  
    type t  
    val porownaj : t -> t -> porownanie  
  end;;
```

Sygnatura kolejki priorytetowej

Example

```
module type PRI_QUEUE_FUNC =  
  functor (Elem : PORZADEK_LINIOWY) ->  
    sig  
      exception EmptyQueue  
      type elem = Elem.t  
      type queue  
      val empty : queue  
      val is_empty : queue -> bool  
      val put : elem -> queue -> queue  
      val getmax : queue -> elem  
      val removemax : queue -> queue  
    end;;
```

Typ queue jest abstrakcyjny, a elem konkretny.
Cokolwiek będzie wiadome o Elem.t będzie też wiadome o elem.
Inaczej nie można włożyć żadnego konkretnego elementu do kolejki.

Sygnatura kolejki priorytetowej

Example

```
module type PRI_QUEUE_FUNC =  
  functor (Elem : PORZADEK_LINIOWY) ->  
    sig  
      exception EmptyQueue  
      type elem = Elem.t  
      type queue  
      val empty : queue  
      val is_empty : queue -> bool  
      val put : elem -> queue -> queue  
      val getmax : queue -> elem  
      val removemax : queue -> queue  
    end;;
```

Typ `queue` jest abstrakcyjny, a `elem` konkretny.
Cokolwiek będzie wiadome o `Elem.t` będzie też wiadome o `elem`.
Inaczej nie można włożyć żadnego konkretnego elementu do kolejki.

Implementacja listowa kolejki priorytetowej

Example

```
module PriQueue : PRI_QUEUE_FUNC =  
  functor (Elem : PORZADEK_LINIOWY) ->  
    struct  
      type elem = Elem.t  
      exception EmptyQueue  
      type queue = elem list  
      let empty = []  
      let is_empty q = q=empty  
      let rec put x q =  
        if q = [] then [x]  
        else if Elem.porownaj x (hd q) = Wiksze then x::q  
        else (hd q)::(put x (tl q))  
      let getmax q =  
        if q=[] then raise EmptyQueue else hd q  
      let removemax (q:queue) =  
        if q=[] then raise EmptyQueue else tl q  
    end;;
```

Kolejka priorytetowa liczb całkowitych

Example

```
module IntOrder =  
  struct  
    type t = int  
    let porownaj (x:int) (y:int) =  
      if x < y then Mniejsze else  
      if x > y then Wieksze else Rowne  
  end;;  
  
module IntQueue = PriorityQueue (IntOrder);;
```

`IntOrder.t` jest typem konkretnym.

Dzięki temu wiadomo, że elementy kolejki `IntQueue` są typu `int`.

Sortowanie sparametryzowane porządkiem liniowym

Example

Porządek liniowy określa kolejność sortowania:

```
module type SORTING =  
  functor (Elem : PORZADEK_LINIOWY) ->  
    sig  
      type elem = Elem.t  
      val sortuj : elem list -> elem list  
    end;;
```

Heap-Sort

Example

Jednym z algorytmów sortowania jest heap-sort:

```
module HeapSort (PrQ : PRI_QUEUE_FUNC): SORTING =  
  functor (Elem : PORZADEK_LINIOWY) ->  
    struct  
      type elem = Elem.t  
      module PQ = PrQ (Elem)  
      open PQ  
      let wkladaj l =  
        List.fold_left (fun h x -> put x h) empty l  
      let rec wyjmuj q a =  
        if is_empty q then a  
        else wyjmuj (removemax q) (getmax q :: a)  
      let sortuj l = wyjmuj (wkladaj l) []  
    end;;
```

Złożenie funktorów

Example

```
module Sortowanie = HeapSort (PriQueue);;  
module S = Sortowanie (IntOrder);;  
S.sortuj [1;7;3;6;5;2;4;8];;  
- : S.elem list = [1; 2; 3; 4; 5; 6; 7; 8]
```

Zapisując moduły w postaci funktorów, tworzymy łatwo wymienne elementy.

Funktory wyższych rzędów

Funktory wyższych rzędów = parametry lub wynik są funktorami.

Example

Mediana: wybór środkowego (w porządku) elementu:

```
module type MEDIANA =  
  functor (Elem : PORZADEK_LINIOWY) ->  
    sig  
      type elem = Elem.t  
      exception PustaLista  
      val mediana : elem list -> elem  
    end;;
```

Rozwiązanie: użyjmy sortowania.

Potrzebujemy: porządku liniowego i algorytmu sortowania.

Funktory wyższych rzędów

Example

```
module Mediana : functor (Sort : SORTING) -> MEDIANA =  
  functor (Sort : SORTING) ->  
    functor (Elem : PORZADEK_LINIOWY) ->  
      struct  
        type elem = Elem.t  
        exception PustaLista  
        module S = Sort (Elem)  
        let mediana l =  
          let s = S.sortuj l  
            and n = List.length l  
          in  
            if n = 0 then raise PustaLista  
            else List.nth s (n / 2)  
      end;;
```

Funktory wyższych rzędów

Example

Mediana to funktor wyższego rzędu.

```
module M = Mediana (HeapSort (PriQueue)) (IntOrder);;  
M.mediana [4;3;5;6;2;1;7];;  
- : M.elem = 4
```