Wstęp do Programowania potok funkcyjny

Marcin Kubica

2010/2011

Outline

Intuicje

Paradygmat programowania imperatywnego:

- program modeluje fragment świata rzeczywistego,
- w świecie rzeczywistym mamy obiekty, które zmieniają swoje stany,
- w programie modelujemy je za pomocą obiektów obliczeniowych, które też mogą zmieniać stany,
- programowanie obiektowe dalej rozszerza tę analogię:
  - klasyfikacja obiektów — hierarchia klas,
  - czynności — metody.

Referencje

Definition (Referencje)

- Instrukcja = wyrażenie typu `unit` + efekty uboczne.
- Referencje to najprostsza imperatywna struktura danych.
- Wskaźnik lub pojemnik, którego zawartość można modyfikować.
- Odpowiada adresowi zmiennej w imperatywnych językach programowania.
- Operacje na referencjach:
  - `(ref) : 'a -> 'a ref`
    `ref` *w* = nowa referencja, zawierająca początkowo *w*,
  - `(!) : 'a ref -> 'a`
    `!`*r* — wyłuskanie wartości,
  - `(:=) : 'a ref -> 'a -> unit`
    r `:=` *w* — zmiana wartości,

Referencje i polimorfizm

- Kłopoty z referencjami i polimorfizmem.
- Referencja wskazuje na wartość przynajmniej tak ogólną jak typ referencji.
- Wyłuskując wskazywaną wartość polimorficzną możemy ją ukonkretnić.
- Przypisując wartość możemy podać wartość bardziej ogólną.
- Przypisując wartość konkretniejszą powodujemy ukonkretnienie typu referencji!
  (Dokładnie zmianę stanu wiedzy kompilatora o typie referencji.)

Referencje i polimorfizm

Example

```
let f x y = x;;
val f : 'a -> 'b -> 'a = <fun>

let g x y = x + 1;;
val g : int -> 'a -> int = <fun>

let h x y = x + y;;
val h : int -> int -> int = <fun>

let i x y = if y then x else 2 * x;;
val i : int -> bool -> int = <fun>

let r = ref g;;
r := f;;
r := h;;                          zmienia się typ r!
r := i;;                                       błąd
```

Konstrukcje imperatywne

Definition

$$\begin{aligned}\langle \texttt{wyrażenie} \rangle ::= \quad & \langle \texttt{wyrażenie} \rangle \{ \ \underline{;} \ \langle \texttt{wyrażenie} \rangle \}^{+} \mid \\ & \underline{\texttt{begin}} \ \langle \texttt{wyrażenie} \rangle \ \underline{\texttt{end}} \mid \\ & \underline{\texttt{if}} \ \langle \texttt{wyrażenie} \rangle \ \underline{\texttt{then}} \ \langle \texttt{wyrażenie} \rangle\end{aligned}$$

- $w_1; w_2; \ldots; w_n$ — sekwencja instrukcji,
- `begin ...end` — nawiasy, blok,
- `if` w `then` x — równoważne `if` w `then` x `else ()`,

Konstrukcje imperatywne

Definition

⟨wyrażenie⟩ ::= while ⟨wyrażenie⟩ do ⟨wyrażenie⟩ done
for ⟨identyfikator⟩ = ⟨wyrażenie⟩ [to | downto] ⟨wyrażenie⟩ do ⟨wyrażenie⟩ done

- `while` w `do ... done`
  Pętla, powtarzamy dopóki $w =$ `true`,
- `for v =` w_1 `to` w_2 `do ... done`
  Pętla, `v` przebiega kolejne wartości całkowite od w_1 do w_2,

Konstrukcje imperatywne

Definition

- Wypisywanie:

```
print_int : int -> unit
print_float : float -> unit
print_string : string -> unit
print_newline : unit -> unit
```

- Wczytywanie:

```
read_int : unit -> int
read_float : unit -> float
read_line : unit -> string
```

Tablice

Definition

Tablice:

- typ `'a array`,
- jednowymiarowe,
- indeksowane liczbami całkowitymi od 0,
- macierz = tablica tablic,
- moduł `Array`.

Tablice

Definition

- `[|`x_1`; ...; `x_n`|]` — konstruktor tablic,
- `make : int -> 'a -> 'a array`
  `make n x` = nowa tablica rozmiaru `n` równa `[|x; ...; x|]`,
- `init : int -> (int -> 'a) -> 'a array`
  `init n f` = nowa tablica rozmiaru `n` równa
  `[|f 0; f 1; ...; f (n-1)|]`,
- `get : 'a array -> int -> 'a`
  `a.(n)` = `get a n` = `n`-ty element tablicy `a`,
  elementy są numerowane od 0 do `length a - 1`,
- `set : 'a array -> int -> 'a -> unit`
  `a.(n) <- x` = `set a n x` — przypisanie `n`-temu elementowi tablicy wartości `x`,

Tablice

Definition

- `length : 'a array -> int`
  `length a` = długość tablicy,
- `make_matrix : int -> int -> 'a -> 'a array array`
  `make_matrix m n x` = macierz, tablica `m`-elementowa tablic `n`-elementowych wypełnionych wartościami `x`,
- `map : ('a -> 'b) -> 'a array -> 'b array`
  analogiczne do `List.map`,
- `fold_left : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a`
  analogiczne do `List.fold_left`,
- `fold_right : ('a -> 'b -> 'b) -> 'a array -> 'b -> 'b`
  analogiczne do `List.fold_right`.

Obiekty i stany

- Jak tworzyć obiekty, które mogą zmieniać stan?
- Najprostszym obiektem, który może zmieniać stan są referencje.
- Można tworzyć obiekty lokalne, dostępne tylko wybranym procedurom.
- Obiekt = procedura + wartości lokalne.

Example

```
let generator s =
  let r = ref s
  in
    function x -> begin
      r := !r + x;
      !r
    end;;
```

- Każde wywołanie tworzy osobną ramkę zawierającą referencję.
- Wynikiem jest procedura, która ma dostęp do tej referencji.

```
let o = generator 0;;
o 22;;
- : int = 22
o 20;;
- : int = 42
```

Konto bankowe

Example

```
let konto pocz =
  let saldo = ref pocz
  in let wplata kwota =
      if kwota > 0 then begin
        saldo := !saldo + kwota;
        !saldo
      end else failwith "Ujemna lub zerowa kwota"
    and wyplata kwota =
      if kwota > 0 then
        if kwota <= !saldo then begin
          saldo := !saldo - kwota;
          !saldo
        end else failwith "Brak środków na koncie"
      else failwith "Ujemna lub zerowa kwota"
    in (wplata, wyplata);;
```

Konto bankowe

Example

```
let (wplac, wyplac) = konto 0;;

wplac 50;;
wyplac 8;;
```

Co by było gdyby konto nie miało argumentu pocz?

```
let konto =
  let saldo = ref 0
  in ...
```

Konto bankowe

Example

```
let (wplac, wyplac) = konto 0;;

wplac 50;;
wyplac 8;;
```

Co by było gdyby `konto` nie miało argumentu `pocz`?

```
let konto =
  let saldo = ref 0
  in ...
```

Metoda Monte Carlo

Definition

- Wynaleziona przez Johna von Neumanna, Stanisława Ulama i Nicholasa Metropolisa, w ramach projektu Manhattan.
- Pierwotnie użyta do badania rozchodzenia się swobodnych neutronów w różnych substancjach.
- Metoda przybliżania probabilistycznego rozmaitych wartości.
- Polega na wielokrotnym losowaniu elementów pewnego zbioru i sprawdzaniu, czy mają określoną własność.
- W ten sposób przybliżamy prawdopodobieństwo, z jakim dane ze zbioru mają daną własność.
- Na podstawie przybliżonego prawdopodobieństwa można przybliżyć szukaną wartość.

Metoda Monte Carlo

Example

Generyczna implementacja metody Monte Carlo:

```
let montecarlo dane wlasnosc liczba_prob =
  let rec montecarlo_rec a n =
    if n = 0 then a
    else if wlasnosc (dane ()) then
      montecarlo_rec (a+1) (n-1)
    else montecarlo_rec a (n-1)
  in
    float (montecarlo_rec 0 liczba_prob) /.
    float (liczba_prob);;
```

Przybliżenie liczby π

Example

Prawdopodobieństwo trafienia w koło jednostkowe $= \frac{\pi}{4}$

```
let square x = x *. x;;

let kolo (x, y) = square x +. square y <= 1.0;;

let kwadrat () =
  (losowa_od_do (-1.0) 1.0, losowa_od_do (-1.0) 1.0);;

let pi = 4.0 *. (montecarlo kwadrat kolo 1000);;
```

Metoda Monte Carlo

Example

```
let losowa_od_do od doo =
  losowa () *. (doo -. od) +. od;;

let losowa () =
  let duzo = 1000
  in let l = float (losowy_int () mod (duzo + 1)) /.
             float (duzo)
     in if l < 0.0 then (-. l) else l;;
```

Metoda Monte Carlo

Example

```
let losowy_int =
  let stan = ref 0
  and a = 937126433
  and b = 937187
  in function () -> (stan := !stan * b + a; !stan);;
```

- Tylko odrobina imperatywności.
- Jak wyglądałby ten program bez imperatywności?

Metoda Monte Carlo

Example

```
let losowy_int =
  let stan = ref 0
  and a = 937126433
  and b = 937187
  in function () -> (stan := !stan * b + a; !stan);;
```

- Tylko odrobina imperatywności.
- Jak wyglądałby ten program bez imperatywności?

Cena programowania imperatywnego

- Kłopoty z pojęciem *tożsamości*:
  - Kiedy x i y to **to samo**?
  - Równość struktur danych =.
  - Tożsamość struktur danych ==.
  - Niezależność struktur danych.
- Źródła błędów programistycznych:
  - Kolejność instrukcji.
  - Zależności między strukturami danych.
  - Efekty uboczne.
  -
- Programowanie współbieżne — różnice stają się jeszcze bardziej widoczne.
- Szczypta imperatywności może uprościć strukturę programu.