Vprašanja za ustni izpit pri Programiranju I

Iztočnice za funkcijsko programiranje

Najpogostejši tipi v OCamlu

okrajšave tipov

- int
- float
- string
- unit
- char
- bool

Zelo pomembna je razlika med tipoma *int* in *float*. Pri *intih* uporabljamo klasične operatorje, medtem, ko za *floate* tem operatorjem dodamo piko na koncu.

tipi funkcije

$$tip_{arg} \rightarrow tip_{rez}$$

tip naborov [1]

$$tip_1 * tip_2 * \cdots * tip_n$$

tip seznama

$$tip_{el} \quad list$$

OCaml je pri tipih v seznamu zelo dosleden. Hkrati so lahko v seznamu zgolj elementi istega tipa.

Vrednostim, ki imajo v tipih spremenljivke, pravimo parametrično polimorfne. *Using parametric polymorphism, a function or data type can be written generically so that it can handle values identically without depending on their type.* [2]

Primerjava med statičnimi in dinamičnimi tipi

OCaml uporablja statične tipe, Python pa dinamične.

statični tipi:

- compiler preveri, da so vsi tipi v funkciji pravi, preden poženemo program
- s tem pridobimo to, da funkcija vedno vrže isti tip ven, kar je zelo uporabno za kompozicije

dinamični tipi:

- programski jezik preveri pravilnost funkcije šele, ko jo kličemo

OCamlovi tipi so bogatejši od Pythonovih, saj nam dajo več podatkov [3]

[1] Prazen nabor je tipa unit().
[2] α tip
[3] Za sezname celih števil nam OCaml vrne tip *int list*, medtem, ko Python vrne zgolj *list*.

Repni klici in repna rekurzija

Klic funkcije je *repen*, če se izvede zadnji. Funkciji, kjer so vsi klici repni, pravimo *repno rekurzivna funkcija*.

OCaml optimizira repne klice.

Python repnih klicev namenoma ne optimizira. [4]

Parametrični tipi in polimorfne funkcije

Funkcije višjega reda in anonimne funkcije

Anonimne funkcije so funkcije, ki jih *ne poimenujemo*. V OCamlu jih zapišemo kar kot *fun* $x \to$ *fun(x)*. Uporabne so predvsem kadar jih uporabimo zgolj enkrat in na seznamu.

```
List.map (fun x -> x * x) [1; 2; 3; 4]
(* - : int list = [1; 4; 9; 16] *)
```

This is handy, as we would otherwise have to name the function first (see let) to be able to use it:

```
let square x = x * x
(* val square : int -> int = <fun> *)

List.map square [1; 2; 3; 4]
(* - : int list = [1; 4; 9; 16] *)
```

Funkcije nam ob dani spremenljivki vrnejo določene vrednosti. Če te vrednosti uporabimo naprej v drugi funkciji, govorimo o *funkcijah višjega reda*. Torej funkcija višjega reda je vsaka funkcija, ki kot argument uporabi neko drugo funkcijo v določeni spremenljivki.

```
let twice f x = f (f x)
(* twice : ('a -> 'a) -> 'a -> 'a *)
```

V OCamlu je navada, da pišemo *fun* $x \to$ *fun(x)*. Operator $\to$ v tem zapisu je desno asociativen.

Delno uporabljene funkcije in curryiranje

Vsote in induktivni tipi

Različice funkcije *fold*

Funkcije *fold* so funkcije višjega reda. Gre za družino funkcij, ki "nabirajo" vrednosti elementov po vrsti. Primer funkcije, ki sodi v družino *fold*, je funkcija max. [5]

[4] PRI OBEH NAPIŠI PREDNOSTI IN SLABOSTI
[5] NAPIŠI KAKO DELUJE

`fold_right`

- vrstni red združevanja elementov je desno → levo
- ni repno rekurzivna

`fold_left`

- vrstni red združevanja elementov je levo → desno
- repno rekurzivna

Tipa funkcij sta različna.

```
# List.fold_left;;
- : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>

# List.fold_right;;
- : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b = <fun>
```

Dokazovanje z indukcijo na seznamih in drevesih

Signature in moduli

Iztočnice za podatkovne strukture in algoritme

Računska zahtevnost

Iskalna drevesa in AVL drevesa

Spremenljive in nespremenljive podatkovne strukture

Predstavitev podatkov v pomnilniku

Razlika med verižnimi seznami in tabelami

Metoda deli in vladaj

Fisher - Yatesov algoritem

Algoritmi za urejanje

Časovna zahtevnost hitrega urejanja

Dinamično programiranje in memoizacija