

Vprašanja za ustni izpit pri Programiranju I

Iztočnice za funkcionalno 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 *floats* tem operatorjem dodamo piko na koncu.

tipi funkcije

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

tip naborov¹

$$tip_1 * tip_2 * \dots * tip_n$$

tip seznama

$$tip_{el} \ list$$

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

¹Prazen nabor je tipa `unit()`.

² α tip

- 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

OCamlov tipi so bogatejši od Pythonovih, saj nam dajo več podatkov³

Repni klici in repna rekurzija

Klic funkcije je *rep*, če se izvede zadnji. Funkciji, kjer so vsi klici repni, pravimo *repno rekurzivna funkcija*. Pri *repno rekurzivnih funkcijah* spreminjaamo samo akumulator, zato si računalniku ni treba zapomniti celotnega procesa ampak zgolj vmesne vrednosti.

OCaml optimizira repne klice.

Python repnih klicev namenoma ne optimizira, da so sporočila o napakah bolj poučna. Tega, da sklad ne raste, ne dosežemo tako, da optimiziramo repno rekurzijo, ampak da optimiziramo repne klice na splošno. Torej poljubna funkcija, ki kliče drugo funkcijo, izgine iz kode. Razlog, zakaj ne želimo po privzetem načinu izvzeti repnih klicev je v tem, da dobimo lep izpis o napakah. Če optimiziramo repne klice in vmesne funkcije izginejo iz zgodbe, se lahko ”izgubimo”.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 → 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] *)
```

³Za sezname celih števil nam OCaml vrne tip *int list*, medtem, ko Python vrne zgolj *list*.

⁴PRI OBEH NAPIŠI PREDNOSTI IN SLABOSTI

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 $fun\ x \rightarrow fun(x)$. Operator \rightarrow 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.

Uporabimo v funkcijah, kjer sledimo vzorcu:

- vzami seznam in začetno vrednost akumulatorja
- sprehodi se čez seznam
- za vsak element v seznamu *naredi nekaj* s trenutno vrednostjo akumulatorja in trenutnega elementa
- nadaljuj z iteracijo po seznamu
- ko prideš do konca seznama, vrni akumulator

Ta postopek lahko skrajšamo s funkcijo *fold*, ki sprejme tri argumente. (*funkcijo f, začetno vrednost akumulatorja in seznam elementov*)

`fold_right`

- vrstni red združevanja elementov je desno \rightarrow levo
- ni repno rekurzivna

`fold_left`

- vrstni red združevanja elementov je levo \rightarrow 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

Indukcija na naravnih številih

$$P(0) \wedge (\forall m.P(m) \Rightarrow P(m^+)) \Rightarrow \forall n.P(n)$$

Indukcija na seznamih

$$P([]) \wedge (\forall x, xs.P(xs) \Rightarrow P(x :: xs)) \Rightarrow \forall ys.P(ys)$$

ideja:

Vsek seznam je ali *prazen* ali pa ima *glavo* in *nek krajši seznam*.⁵ Če imamo predikat P , ki velja na praznem seznamu in velja, da iz tega, ko P velja na repu xs , P velja tudi na glavi in repu $x :: xs$, potem predikat P velja na celiem seznamu.

Ne moremo reči, da P velja za x , saj je P predikat, ki velja na seznamih, x pa je zgolj element seznama, zato je nesmiselno govoriti o $P(x)$.

Indukcija na seznamih deluje, ker jo lahko prevedemo na indukcijo na naravnih številih. Iz predikata P na seznamih moramo predelati predikat Q na naravnih številih. To storimo tako, da definiramo

$$Q(n) := \forall ws.|ws| = n \Rightarrow P(ws).$$

Z besedami to pomeni " $Q(n)$ bo veljalo, če za vse sezname dolžine n velja $P(ws)$." V tem primeru je $Q(0)$ ekvivalentno $P([])$. To, da gremo iz tega, da če velja za rep, velja tudi za glavo, je ravno korak indukcije na naravnih številih. Mi predpostavimo, da velja za sezname ...⁶.

Indukcija na drevesih

$$P(\text{Empty}) \wedge (\forall x, l, d.P(l) \wedge P(d) \Rightarrow P(\text{Node}(l, x, d))) \Rightarrow \forall t.P(t)$$

Najprej pokažemo, da predikat velja za prazno drevo. V indukcijskem koraku predpostavimo, da velja za *levega in desnega otroka*. Razlika od indukcije na dobro urejenih množicah je v tem, da smo tam predpostavili, da velja za *vse manjše*. Tu predpostavimo, da velja samo za dve *direktno manjši drevesi*. Pomembno pri indukciji na drevesih je to, da imamo neko *velikost*, nekaj, s čimer lahko indukcijo na drevesih prevedemo na indukcijo na naravnih številih. Dober primer je *globina*.

⁵Ignoriramo neskončne sezname.

⁶DOPÍŠI

Indukcija na poljubnem tipu

```
type nek_moj_tip =
| A
| B of nek_moj_tip
```

1. pokažemo, da predikat P velja na A
2. pokažemo, da iz tega, da velja za "*nek_moj_tip*", velja tudi za B

$$P(A) \wedge \forall m.P(m) \Rightarrow P(Bm) \Rightarrow \forall n.P(n)$$

Indukcija za bool

$$P(A) \wedge P(B) \Rightarrow \forall b.P(b)^7$$

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 seznamami in tabelami

Metoda deli in vladaj

Fisher - Yatesov algoritem

Algoritmi za urejanje

Časovna zahtevnost hitrega urejanja

Dinamično programiranje in memoizacija

⁷Predpostavki sta samo, da velja za A in za B . Nimamo nobenih manjših problemov, s katerimi bi prevedli na "navadno" indukcijo.