Haskell in razreditipov

prejšnjič...

Fleksibilnost tipov lahko zagotovimo na več načinov

parametrični polimorfizem

generiki (Java, Rust), polimorfizem (OCaml, Haskell), ...

ad-hoc polimorfizem

značilnosti/traits (Rust), razredi tipov (Haskell), ...

podtipi

implicitne pretvorbe, moduli (OCaml), podrazredi, ...

Vrednosti **manjšega** tipa so kompatibilne z **večjim**

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash e : B} \leq B$$

Spoznali smo pravila za določanje podtipov

$$A_1 \le B_1$$
 $A_2 \le B_2$
 $A_1 \times A_2 \le B_1 \times B_2$

$$B_1 \le A_1 \qquad A_2 \le B_2$$

$$A_1 \to A_2 \le B_1 \to B_2$$

Zapisni tipi imajo bogato strukturo podtipov

$$\forall i \leq m . \exists j \leq n . \ell'_i = \ell_j \land A_j \leq B_i$$

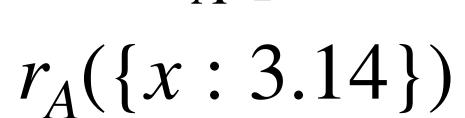
$$\{\ell_1: A_1; ...; \ell_n: A_n\} \leq \{\ell_1': B_1; ...; \ell_m': B_m'\}$$

Spremenljivi zapisi podtipov ne podpirajo

$$A = \{ \text{mutable } x : \text{int} \}$$
 $B = \{ \text{mutable } x : \text{float} \}$

branje polja

let
$$r_A(p:A) = p.x + 10$$





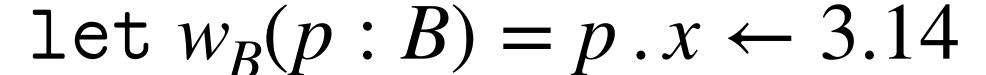
let
$$r_B(p:B) = p.x + 3.14$$

$$r_B(\{x:10\})$$

pisanje v polje

let
$$w_A(p:A) = p.x \leftarrow 10$$

$$w_A(\{x:3.14\})$$



$$w_B(\{x:10\})$$



Objekti so bolj ali manj rekurzivni zapisi

konstruktorji & metode

dedovanje

prekrivanje (overriding)

preobteževanje (overloading)

generične metode & razredi

abstraktne/virtualne metode & razredi

vmesniki

enkapsulacija

funkcije

nominalni podtipi

senčenje

ad-hoc polimorfizem

parametrični polimorfizem

specifikacije

specifikacije

modularnost & abstrakcija

Pri razredih imamo dve pristopa k podtipom

```
class C { int x; }
class D { int x; float y; }
class E extends C { float y; }
```

nominalni pristop

Podrazredi so tisti, ki jih programer **navede**.

$$E \leq C \qquad D \nleq E$$

strukturni pristop

Podrazredi so vsi s kompatibilno strukturo.

$$E \leq C$$
 $D \leq E \leq D$

nominalni in strukturni W OCamlu

Uvod v Haskell



Razreditipov

prihodnjič...

Posvetili se bomo računskim učinkom

program

funkcija + učinki

Spoznali bomo monade

```
T:: Type -> Type
return :: a -> T a
  >>= :: T a -> (a -> T b) -> T b
```