

Podtipi in objekti

prejšnjič...

Videli smo različne **enakosti**, vgrajene v Prologu

identiteta z združevanjem

```
?- X + 2 = 1 + 2.  
X = 1.
```

```
?- X + 2 = 2 + 1.  
false.
```

prirejanje

```
?- X is 1 + 2.  
X = 3.
```

```
?- 1 + 2 is 1 + 2.  
false.
```

identiteta brez združevanja

```
?- X + 2 == 1 + 2.  
false.
```

```
?- X = 1, X + 2 == 1 + 2.  
X = 1.
```

enakost

```
?- 1 + 2 =:= 3.  
true.
```

```
?- X + 2 =:= 1 + 2.  
ERROR: Arguments are not  
sufficiently instantiated
```

Spoznali smo logično programiranje z omejitvami

```
?- 1 #=< P, P #=< Q, P * Q #= 12.
```

```
P in 1..12, Q #>= P,  
P * Q #= 12, Q in 1..12.
```

```
?- 1 #=< P, P #=< Q, P * Q #= 12, label([P,Q]).
```

```
P = 1, Q = 12; P = 2, Q = 6; P = 3, Q = 4 .
```

Poleg aritmetičnih smo spoznali tudi **globalne omejitve**

```
permutacija(N, P) :-  
    length(P, N),  
    P ins 1..N,  
    all_distinct(P).
```

```
?- permutacija(3, P), label(P).
```

```
P = [1, 2, 3]; P = [1, 3, 2]; P = [2, 1, 3];  
P = [2, 3, 1]; P = [3, 1, 2]; P = [3, 2, 1].
```

Tipi & podtipi

Veljavne tipe izrazov določimo induktivno

$$\Gamma \vdash e : A$$

$$\frac{}{\Gamma \vdash \text{true} : \text{bool}}$$

$$\frac{}{\Gamma \vdash \text{false} : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : A \quad \Gamma \vdash e_2 : B}{\Gamma \vdash (e_1, e_2) : A \times B}$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}}$$

$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash e_1 : A \quad \Gamma \vdash e_2 : A}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : A}$$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x. e : A \rightarrow B}$$

$$\frac{\Gamma \vdash e_1 : A \rightarrow B \quad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1 e_2 : B}$$

Tipi zagotavljajo **varnost** pri izvajanju

napredek

Izraz s tipom je **vrednost** ali pa lahko **naredi korak**.

$$\vdash e : A \implies (\exists v. e = v) \vee (\exists e'. e \mapsto e')$$

ohranitev

Na vsakem koraku se tip **ohrani**.

$$(\vdash e : A \wedge e \mapsto e') \implies \vdash e' : A$$

Izpeljava in veljavnost tipov sta povezani

skladnost

Izpeljani tip je **veljaven**.

$$(\vdash e \Rightarrow A) \implies (\vdash e : A)$$

polnost

Izpeljemo lahko **najbolj splošen** tip.

$$(\vdash e : A) \implies \exists A', \sigma. (\vdash e \Rightarrow A') \wedge (A = \sigma(A'))$$

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**

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

Podtipi na osnovnih tipih se med jeziki razlikujejo

`int < float float < int`

C		
JavaScript		
Java		
Python		
OCaml		
Haskell		

Na podtipih velja šibka urejenost

$$\overline{A \leq A}$$

$$\frac{A_1 \leq A_2 \quad A_2 \leq A_3}{A_1 \leq A_3}$$

Pri produktih in vsotah imamo podtipe **v globino**

$$\frac{A_1 \leq B_1 \quad A_2 \leq B_2}{A_1 \times A_2 \leq B_1 \times B_2}$$

$$\frac{A_1 \leq B_1 \quad A_2 \leq B_2}{A_1 + A_2 \leq B_1 + B_2}$$

$$\overline{A_1 \rightarrow A_2 \leq B_1 \rightarrow B_2}^{???, ???}$$

Funkcijski tipi so v domeni **kontravariantni**

$$\frac{B_1 \leq A_1 \quad A_2 \leq B_2}{A_1 \rightarrow A_2 \leq B_1 \rightarrow B_2}$$

Zapisi

Zapisi so poimenovani produkti

Python razredi / ~slovarji / NamedTuple

C strukture / *structs*

Java ~ razredi / Record (Java 14)

OCaml zapisi / *record*

Haskell zapisi / *record*

JavaScript ~ objekti

SQL vrstice

Zapisni tip je sestavljen iz tipov polj

$$\frac{e_1 : A_1 \quad \dots \quad e_n : A_n}{\{\ell_1 = e_1; \dots; \ell_n = e_n\} : \{\ell_1 : A_1; \dots; \ell_n : A_n\}}$$

Tudi pri zapisih imamo podtipe v globino

$$\frac{A_1 \leq B_1 \quad \dots \quad A_n \leq B_n}{\{\ell_1 : A_1; \dots; \ell_n : A_n\} \leq \{\ell_1 : B_1; \dots; \ell_n : B_n\}}$$

Pri zapisih imamo podtipe tudi v širino

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

$$\{\ell_1 : A_1; \dots; \ell_n : A_n\} \leq \{\ell'_1 : B_1; \dots; \ell'_m : B'_m\}$$

$\{x : \text{int}; y : \text{float}; z : \text{float}\}$ $\leq ?$ $\{x : \text{float}; y : \text{float}\}$

$\{x : \text{float}\} \leq^? \text{float}$ $\text{float} \leq^? \{x : \text{float}\}$

$$\{a : \{x : \text{int}; y : \text{float}\}; b : \{u : \text{int}\}\}$$
$$\leq^?$$
$$\{a : \{y : \text{float}\}; b : \{\}\}$$

$$\{x : \text{int}\} \rightarrow \{y : \{u : \text{float}\}; z : \text{int}\}$$
$$\leq ?$$
$$\{x : \text{int}; y : \text{int}\} \rightarrow \{y : \{\}; z : \text{float}\}$$

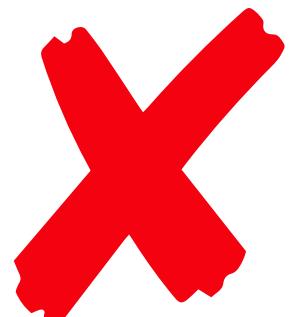
Spremenljivi zapisi podtipov ne podpirajo

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

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

branje polja

let $r_A(p : A) = p.x + 10$
 $r_A(\{x : 3.14\})$



pisanje v polje

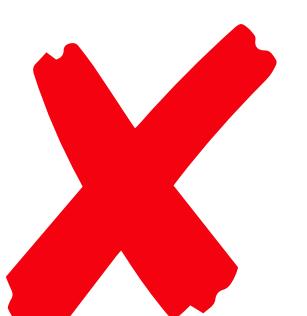
let $w_A(p : A) = p.x \leftarrow 10$
 $w_A(\{x : 3.14\})$



let $r_B(p : B) = p.x + 3.14$
 $r_B(\{x : 10\})$



let $w_B(p : B) = p.x \leftarrow 3.14$
 $w_B(\{x : 10\})$



Objekti

Objekti so zapisi, ki se sklicujejo nase

```
{  
    x = 3.0;  
    y = 4.0;  
  
    dst = λp . √(this.x - p.x)2 + (this.y - p.y)2  
}
```

objekt = zapis + rekurzija

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

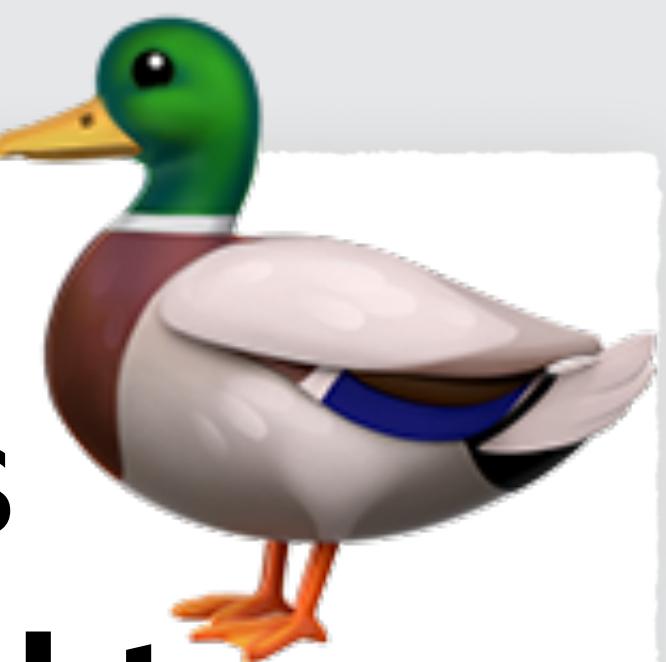
Podrazredi so tisti, ki jih programer **navede**.

$$E \leq C \quad D \not\leq E$$

strukturni pristop

Podrazredi so vsi s **kompatibilno strukturo**.

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



nominalni i strukturni
podtypi
w ocamlu

Z objektnim programiranjem povezujemo še več konceptov

konstruktorji & metode

funkcije

dedovanje

nominalni podtipi

prekrivanje (overriding)

senčenje

preobteževanje (overloading)

ad-hoc polimorfizem

generične metode & razredi

parametrični polimorfizem

abstraktne/virtualne metode & razredi

specifikacije

vmesniki

specifikacije

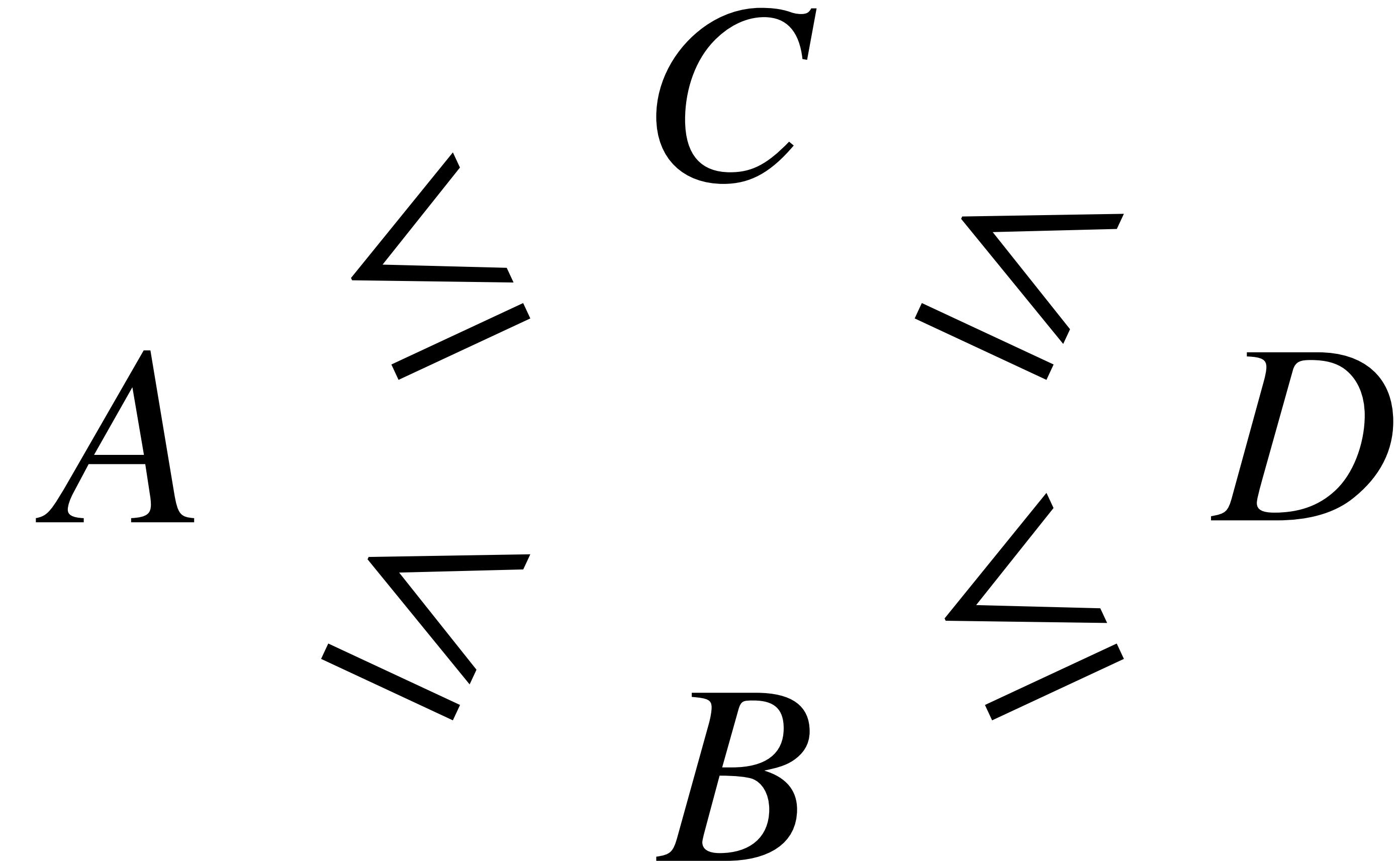
enkapsulacija

modularnost & abstrakcija



objektiv
vocam

Podtipi morajo zadoščati **koherentnosti**



prihodnjič...

Vrnili se bomo k jeziku Haskell



Ad-hoc polimorfizem bomo organizirali z razredi tipov

```
class Num a where
    (+) :: a -> a -> a
    (-) :: a -> a -> a
    (*) :: a -> a -> a
    negate :: a -> a
    abs :: a -> a
    signum :: a -> a
    fromInteger :: Integer -> a
```