Koinduktivni tipi & izpeljava tipov

Induktivni: končni seznami,
končna drevesa

Koinduktivni tipi:

- končne in neskončne strukture
- leni seznami: končni ali neskončni
- tok podatkov
- input/output

```
type list = Nil | Cons of int * list          OCaml
data List = Nil | Cons Int List              Haskell
```

Induktivni seznami so končni.

Koinduktivni seznami: končni in neskončni.

Tok podatkov tipa α:

- tok je urejeni par (d, t), kjer je $d : \alpha$

podatek in t je tok

type stream = Stream of int * stream

↳ induktivno: končnen tok Stream(3, Stream(8, ...))

ne moremo se ustaviti

končnega toka ni!

odgovor: ta tip je prazen

koinduktivno: ✓

Normalni ljudje: List Int seznam celih števil

Ocaml: int list

Normalni ljudje: $\mathbb{R} \times \mathbb{R}$ $(3, -5)$, List Int, int list

Haskell: $(\mathbb{R}, \mathbb{R})$ $(3, -5)$

$(\mathbb{R}, \mathbb{R}, \mathbb{R})$ urejene trojice

$[\mathbb{R}]$ tip seznamov realnih števil

$[3, \sqrt{2}, \pi]$

Eratostenovo sito

2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11

Asistent pri Analizi

$$f\,x = \begin{cases} e_1 & \text{če } p_1 \\ e_2 & \text{če } p_2 \\ e_3 & \text{sicer} \end{cases}$$

Haskell

```
f x | p1 = e1
    | p2 = e2
    | otherwise = e3
```

True

```
f x = if p1 then e1 else (if p2 then e2 else e3)
```

```
h x (f (g x y))  ⇒  h x $ f $ g x y
```

```
0 : 1 : 2 : 3
↑
```

Zavlačevanje (thunking):

izraz $e : T$ ← recimo, da se zacikla

$\underbrace{\lambda().\, e}_{\text{unit} \to T}$ ← se ne zaciklа

Zavlačevanje (thunk): $\underbrace{\lambda().\, e}_{t}$

Sprožimo (force): $t()$

Java: (približno)

```
public static T t() { return e; }
```

Sprožimo s klicem `t()`

$$f(x) = f(x_0) + f'(x_0)\cdot(x-x_0) + \cdots + \frac{f^{(n)}(x_0)}{n!}\cdot(x-x_0)^n + \cdots$$

$$\sin x = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 \pm \cdots$$

↑

sin x lahko predstavimo s tokom koeficientov v potenčni vrsti

$$[0, 1, 0, -\frac{1}{3!}, 0, \frac{1}{5!}, 0, \dots]$$
$$x^0 \quad x^1 \quad x^2 \quad x^3$$

I/O

Program, ki izvaja operacijo na input/output:

- izračuna vrednost (tipa a) (brez I/O) in jo vrne
- pošlje podatek na output in nadaljuje
- prebere podatek z input in nadaljuje v odvisnosti od podatka

```
IO a =
  | Return a
  | Write String (IO a)
  | Read (String -> IO a)
```

Izpeljava tipov

Preverjanje tipov:

- programer zapiše tip izraza
- prog. jezik preveri, da izraz ima dani tip

Izpeljava:

- prog. jezik sam ugotovi tip izraza

```
if (i<7) {          ↗ preveri bool
    int a = 3+8;    podano tip, se preveri
    var b = 3+8     izpelji tip b
```

3+8

```
var c = o.mojaMetoda(3);
                ↳ vrne tip T
```

Primer:

```
fun x -> x + 3
```

A (tip x); int (tip x + 3)

x: int? → A = int

3: int? ✓

: A → int

A = int

odgovor: int → int

```
if 3 < 5 then (fun x -> x) else (fun y -> y + 3)
```

3 : int? ✓, 5 : int? ✓, $3 < 5$: bool

x : α, (fun x -> x) : $\alpha \to \alpha$

y : β, y : int?, 3 : int? ✓, $y + 3$: int

$\beta = int$

(fun y -> y + 3) : $\beta \to int$

$\alpha \to \alpha = \beta \to int$

$\alpha \to \alpha = int \to int$

$\alpha = int$

odgovor: $int \to int$