Tipi

prejšnjič...

Rekurzivne definicije smo izenačili z **negibnimi točkami**

$$f = \lambda n \,.\, \texttt{if}\; n = 0 \;\texttt{then}\; 1 \;\texttt{else}\; n \cdot f(n-1)$$

$$\Phi = \lambda g \,.\, \Big(\lambda n \,.\, \texttt{if}\; n = 0 \;\texttt{then}\; 1 \;\texttt{else}\; n \cdot g(n-1)\Big)$$

$$f = \Phi f$$

Videli smo, kako rekurzivne funkcije dobimo z **iteracijo**

$$\Phi = \lambda g\,.\,\lambda n\,.\,\texttt{if}\; n = 0 \;\texttt{then}\; 1 \;\texttt{else}\; n \cdot g(n-1)$$

$$f_0$$
$$\Phi(f_0)$$
$$\Phi(\Phi(f_0))$$
$$\Phi(\Phi(\Phi(f_0)))$$

Spoznali smo vsestransko uporabne **rekurzivne tipe**

$$e ::= n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 / e_2 \mid -e$$

```
type expression =
   | Numeral of int
   | Plus of expression * expression
   | Minus of expression * expression
   | Times of expression * expression
   | Divide of expression * expression
   | Negate of expression
```

S **koinduktivnimi** tipi smo predstavili **neskončne** podatkovne strukture

```
type seznam = Nil | Cons of int * seznam
```

```
Cons (1, Cons (2, Cons (3, Cons (4, ...))))
```

ne

da

induktivni tipi

koinduktivni tipi

praštevila

v Haskellu

Sistemi tipov

Programski jeziki imajo različno **krepke** sisteme tipov

OCaml

```
# (42, "foo", false)
- : int * string * bool = ...
```

Python

```
>>> type((42, 'foo', False))
<class 'tuple'>
```

strojna koda

```
0x2a 0x2a 0x00
```

Statični tipi se preverijo med prevajanjem, **dinamični** med izvajanjem

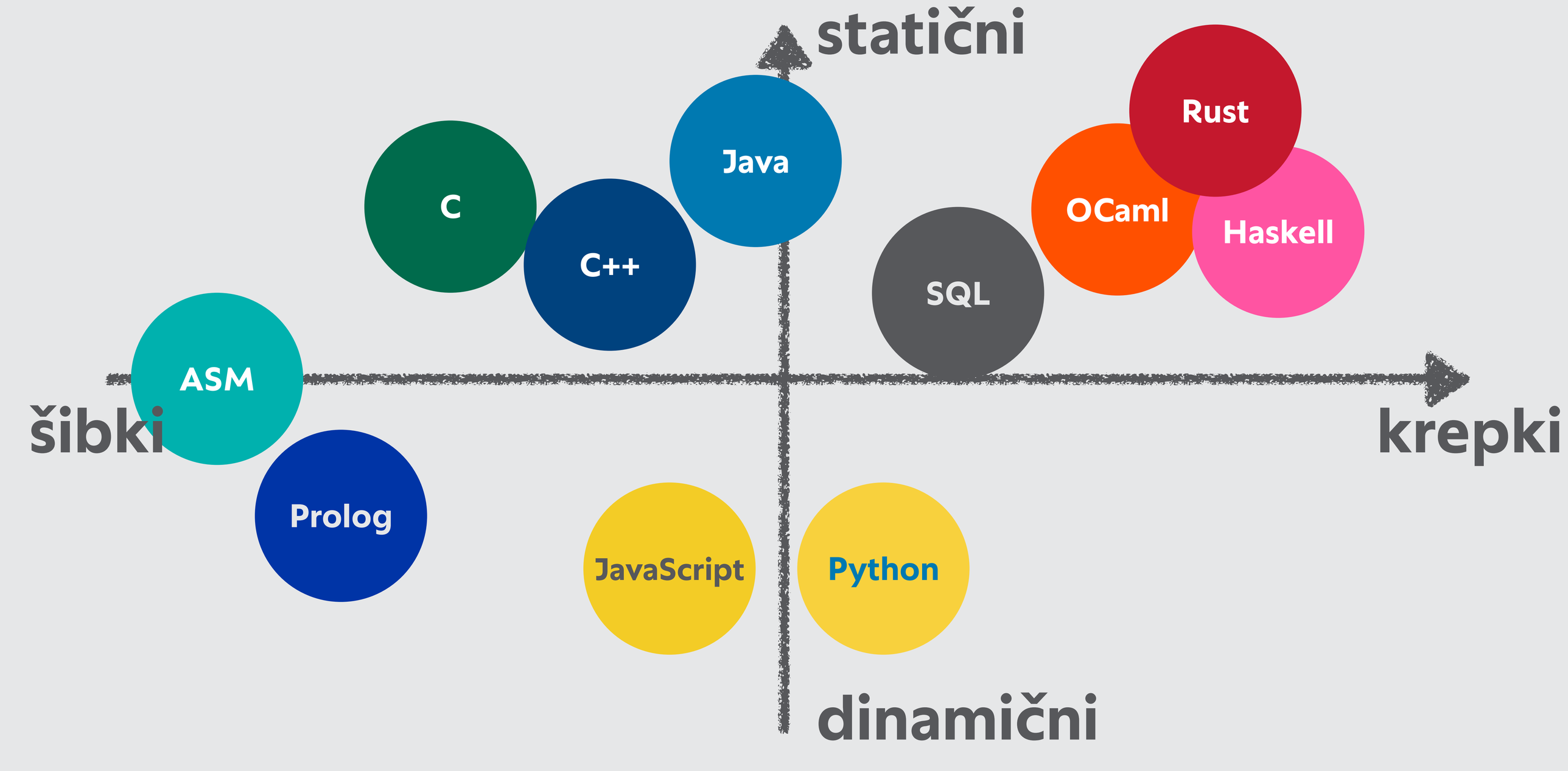
OCaml

```
# let je_majhen x =
    if x < 10 then "Da" else false

Error: This expression has type bool but an expression was expected of
type string
```

Python

```
>>> def je_majhen(x):
...     return "Da" if x < 10 else False
>>> je_majhen(5) + "!"
"Da!"
>>> je_majhen(15) + "!"

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'bool' and 'str'
```

Različni **sistemi** tipov služijo različnim **namenom**

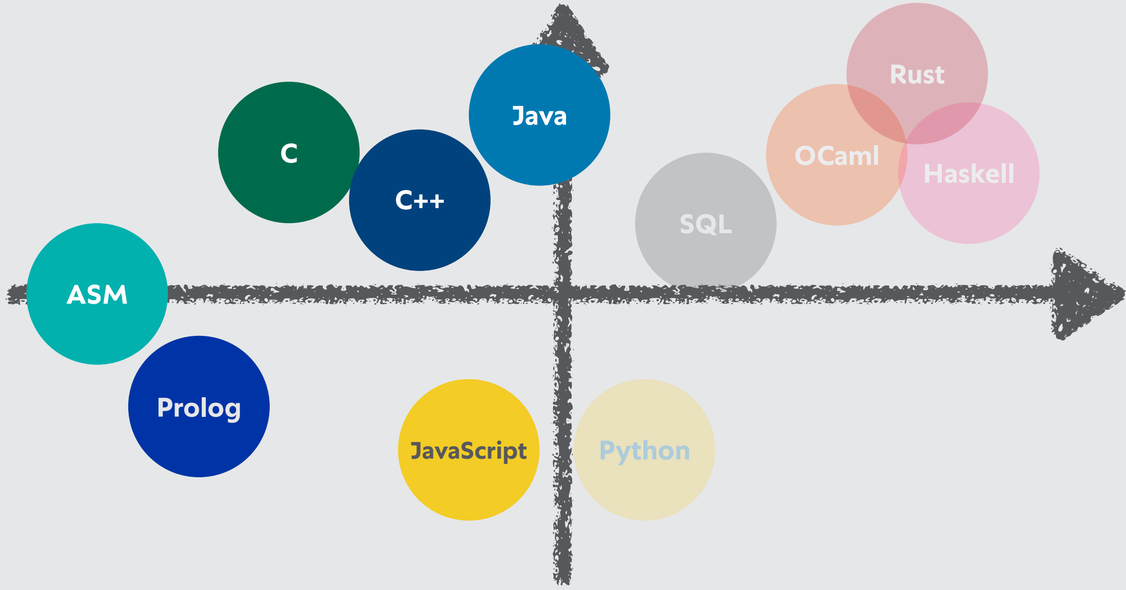


Šibkejši tipi nudijo večjo **fleksibilnost**

```
10 + 3.14
```

```
10 + false
```

```
10 + "foo"
```

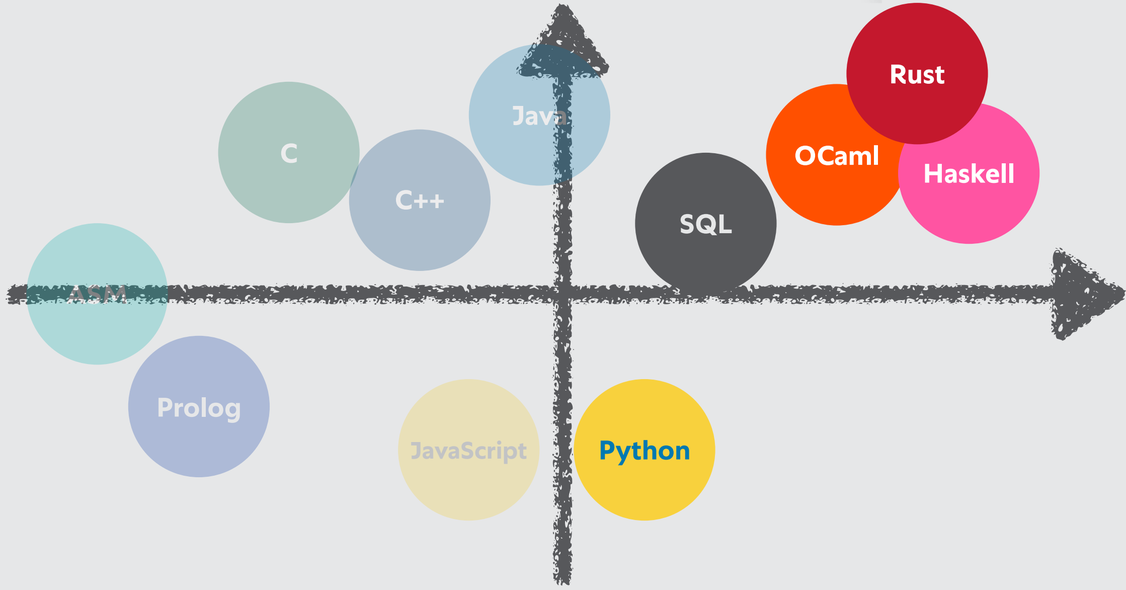



```
if "abc".isdigit:
    print("abc je številka")
```

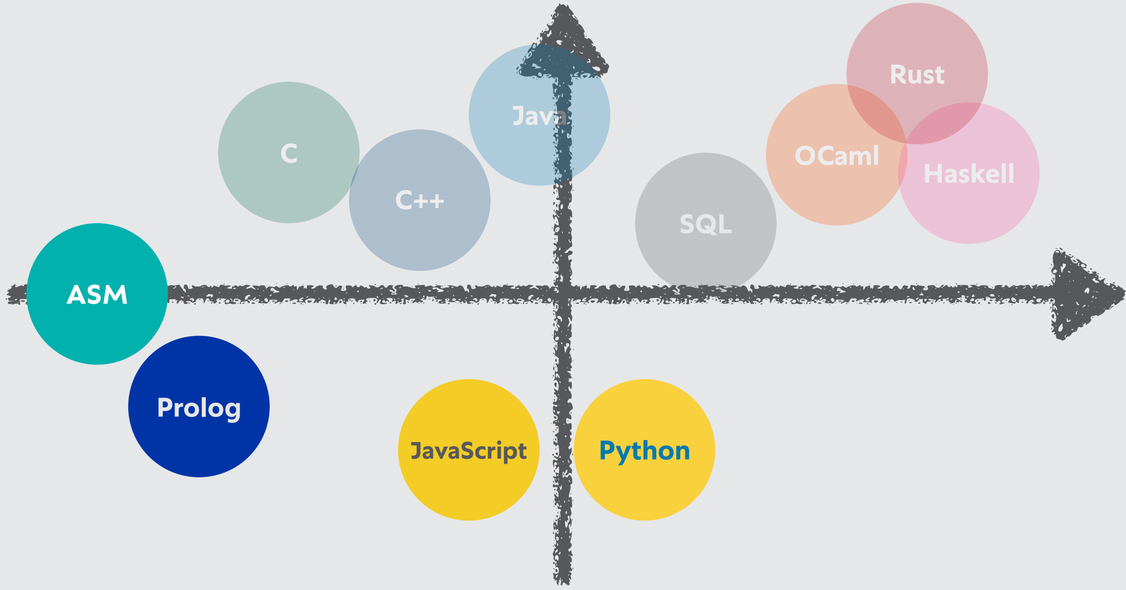
Krepkejši tipi zagotavljajo **varnost** med izvajanjem

$$(\vdash e : \tau \ \wedge \ e \twoheadrightarrow e') \implies \vdash e' : \tau$$

$$\vdash e : \tau \implies e = v \vee \exists e' . e \twoheadrightarrow e'$$



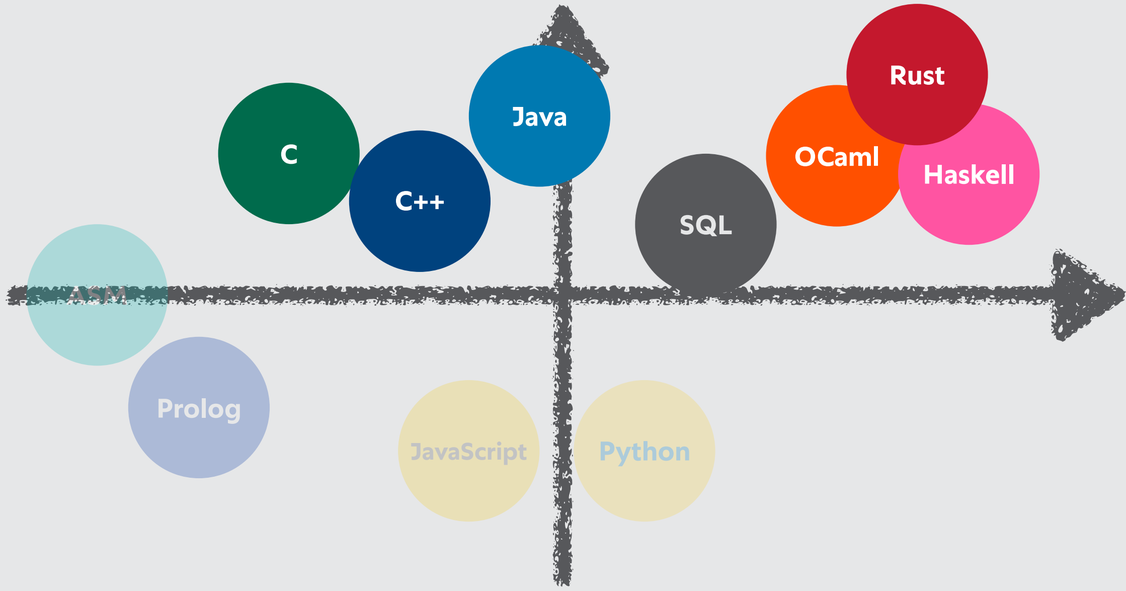
Dinamični tipi *(na začetku)* omogočajo hitrejše **pisanje**



Statični tipi *(med drugim)* omogočajo hitrejše **izvajanje**

```
[(42,true),(5,false),(1,false)]
```

```
[42,true,5,false]
```



Sistemi tipov se razlikujejo tudi po podpori **polimorfizmu**

	parametrični	**ad-hoc**
primeri	`compose, length, map, reverse`	`+, <, toString`
Java	generiki	preobteževanje
OCaml	`('a -> 'b) -> 'a list -> 'b list`	❌
Haskell	`(a -> b) -> List a -> List b`	razredi tipov *(konec leta)*
C	😬	😬

Izpeljava tipov

Tipe lahko jezik **preveri** ali **izpelje**

Java

```
public static <A, B> List<B> map(List<A> lst, Function<A, B> f) {
    ...
}
```

OCaml

```
let rec map f = function
  | [] -> []
  | hd :: tl -> f hd :: map f tl

val map : ('a -> 'b) -> 'a list -> 'b list
```

Izpeljavo in **preverjanje** lahko kombiniramo

brez anotacij

```
let rec map f = function
  | [] -> []
  | hd :: tl -> f hd :: map f tl

val map : ('a -> 'b) -> 'a list -> 'b list
```

z anotacijami

```
let rec map (f : int -> string) = function
  | [] -> []
  | hd :: tl -> f hd :: map f tl

val map : (int -> string) -> int list -> string list
```

Pri izpeljavi želimo najti **glavni tip** programa

$$\lambda x \,.\, \lambda y \,.\, x$$

$\texttt{int} \to \texttt{int} \to \texttt{int}$

$\texttt{int} \to \texttt{bool} \to \texttt{int}$

$\alpha \to \alpha \to \alpha$

$\alpha \to \beta \to \alpha$

Hindley-Milnerjev algoritem poteka v dveh fazah

$$\lambda f\,.\,f(f\,0)$$

1. nastavimo **tip & enačbe**

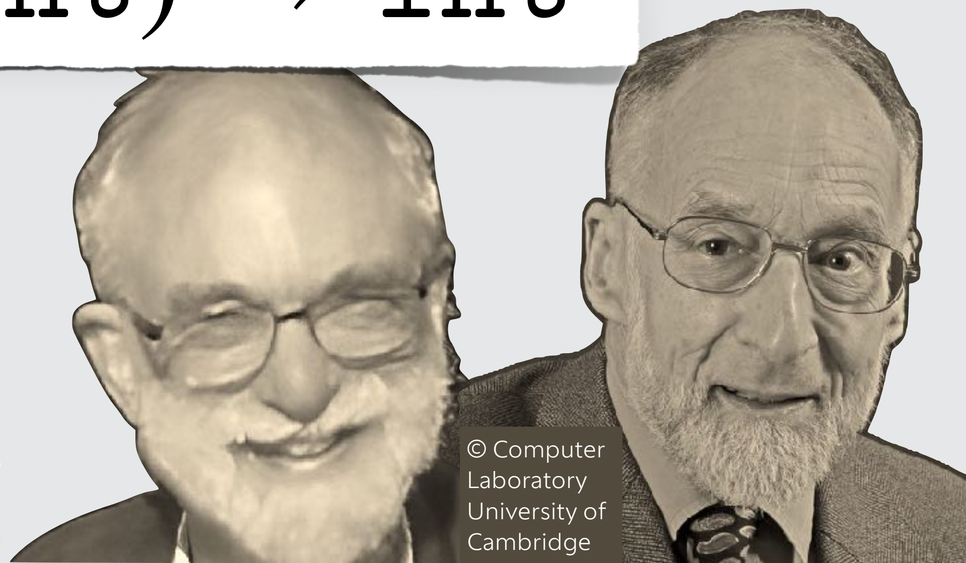
$$\begin{aligned}\alpha &= \gamma \to \beta\\ \gamma &= \texttt{int}\\ \gamma &= \beta\end{aligned}$$

2. rešimo enačbe

$$\begin{aligned}\alpha &\mapsto (\texttt{int} \to \texttt{int})\\ \beta &\mapsto \texttt{int}\\ \gamma &\mapsto \texttt{int}\end{aligned}$$

vstavimo rešitev v tip

$$(\texttt{int} \to \texttt{int}) \to \texttt{int}$$



Roger Hindley, 1939–

Robin Milner, 1934–2010

V prvi fazi **nastavimo** tip ter **enačbe** med neznankami

$x : \texttt{bool}, y : \texttt{int}, z : \alpha$

Tip **konstant** določimo enostavno

$$\Gamma \vdash \texttt{true} \Rightarrow \texttt{bool} \mid \emptyset$$

$$\Gamma \vdash \texttt{false} \Rightarrow \texttt{bool} \mid \emptyset$$

$$\Gamma \vdash n \Rightarrow \texttt{int} \mid \emptyset$$

Tip **spremenljivk** preberemo iz konteksta

$$\Gamma \vdash x \Rightarrow \Gamma(x) \mid \varnothing$$

Pri **aritmetičnih/logičnih izrazih** dobimo enačbe za podizraze

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_1 \mid E_1 \qquad \Gamma \vdash e_2 \Rightarrow \tau_2 \mid E_2}{\Gamma \vdash e_1 + e_2 \Rightarrow \texttt{int} \mid E_1, E_2, \tau_1 = \texttt{int}, \tau_2 = \texttt{int}}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_1 \mid E_1 \qquad \Gamma \vdash e_2 \Rightarrow \tau_2 \mid E_2}{\Gamma \vdash e_1 < e_2 \Rightarrow \texttt{bool} \mid E_1, E_2, \tau_1 = \texttt{int}, \tau_2 = \texttt{int}}$$

$$\Gamma \vdash x < y + 1 \Rightarrow \tau \mid E$$

Izpeljave bomo pisali **na krajše**

$$\dfrac{\dfrac{}{\Gamma \vdash x \Rightarrow \alpha \mid \varnothing} \qquad \dfrac{\dfrac{}{\Gamma \vdash y \Rightarrow \beta \mid \varnothing} \quad \dfrac{}{\Gamma \vdash 1 \Rightarrow \texttt{int} \mid \varnothing}}{\Gamma \vdash y + 1 \Rightarrow \texttt{int} \mid \beta = \texttt{int}, \texttt{int} = \texttt{int}}}{\Gamma \vdash x < y + 1 \Rightarrow \texttt{bool} \mid \beta = \texttt{int}, \texttt{int} = \texttt{int}, \alpha = \texttt{int}, \texttt{int} = \texttt{int}}$$

$$\overbrace{\underbrace{x}_{\alpha} < \overbrace{\underbrace{y}_{\beta} + \underbrace{1}_{\texttt{int}}}^{\texttt{int}}}^{\texttt{bool}}$$

$$\beta = \texttt{int}$$

$$\texttt{int} = \texttt{int}$$

$$\alpha = \texttt{int}$$

$$\texttt{int} = \texttt{int}$$

Pri **pogojnem** stavku se morata tipa vej ujemati

$$\frac{\Gamma \vdash e \Rightarrow \tau \mid E \quad \Gamma \vdash e_1 \Rightarrow \tau_1 \mid E_1 \quad \Gamma \vdash e_2 \Rightarrow \tau_2 \mid E_2}{\Gamma \vdash \texttt{if}\, e\, \texttt{then}\, e_1\, \texttt{else}\, e_2 \Rightarrow \tau_1 \mid E, E_1, E_2, \tau = \texttt{bool}, \tau_1 = \tau_2}$$

Pri **parih** združimo tipe in enačbe podizrazov

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_1 \mid E_1 \qquad \Gamma \vdash e_2 \Rightarrow \tau_2 \mid E_2}{\Gamma \vdash (e_1, e_2) \Rightarrow \tau_1 \times \tau_2 \mid E_1, E_2}$$

Pri **projekcijah** je ena komponenta tipa nedoločena

$$\frac{\Gamma \vdash e \Rightarrow \tau \mid E}{\Gamma \vdash \texttt{fst}\, e \Rightarrow \alpha \mid E, \tau = \alpha \times \beta}$$

$$\frac{\Gamma \vdash e \Rightarrow \tau \mid E}{\Gamma \vdash \texttt{snd}\, e \Rightarrow \beta \mid E, \tau = \alpha \times \beta}$$

Pri **funkcijah** kontekst razširimo s svežim parametrom

$$\frac{\Gamma, x : \alpha \vdash e \Rightarrow \tau \mid E}{\Gamma \vdash \lambda x\,.\,e \Rightarrow \alpha \to \tau \mid E}$$

$$\lambda b\, x\, .\, \texttt{if}\ b\ \texttt{then}\ x\ \texttt{else}\ 0$$

Pri **aplikaciji** mora biti tip prvega podizraza funkcijski

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_1 \mid E_1 \qquad \Gamma \vdash e_2 \Rightarrow \tau_2 \mid E_2}{\Gamma \vdash e_1\, e_2 \Rightarrow \alpha \mid E_1, E_2, \tau_1 = \tau_2 \to \alpha}$$


λf x . f(f x)

$$\lambda p\, f\, x\, .\, \texttt{if}\ p\ x\ \texttt{then}\ f\ x\ \texttt{else}\ x$$

Pri **rekurzivnih definicijah** nastavimo rekurzivno enačbo

$$\frac{\Gamma, f : \alpha \vdash e \Rightarrow \tau \mid E}{\Gamma \vdash (\texttt{rec}\, f\, .\, e) \Rightarrow \tau \mid E, \alpha = \tau}$$

$$f = \lambda n\,.$$

$$\texttt{if}\ n = 0\ \texttt{then}\ 1$$

$$\texttt{else}\ n * f(n - 1)$$

nastavljanje enačb v jeziku poly

V drugi fazi iz enačb postopoma dobimo **rešitev**

$$\alpha = \texttt{int}, \beta = \alpha \to \alpha$$

$$\rightsquigarrow \cdots \rightsquigarrow$$

$$\alpha \mapsto \texttt{int}, \beta \mapsto \texttt{int} \to \texttt{int}$$

Enačbe postopoma **združujemo**, dokler jih ne zmanjka

enačbe rešitve

$$\langle E \mid R \rangle \rightsquigarrow \langle E' \mid R' \rangle$$

$$\langle E \mid \varnothing \rangle \rightsquigarrow \cdots \rightsquigarrow \langle \varnothing \mid R' \rangle$$

Enačbe, kjer se **obe strani ujemata**, lahko zavržemo

$$\langle \sigma = \sigma, E \mid R \rangle$$

$$\rightsquigarrow$$

$$\langle E \mid R \rangle$$

Enačbo med **produktoma razbijemo** na dve enačbi

$$\langle \sigma_1 \times \tau_1 = \sigma_2 \times \tau_2, E \mid R \rangle$$

$$\rightsquigarrow$$

$$\langle \sigma_1 = \sigma_2, \tau_1 = \tau_2, E \mid R \rangle$$

Podobno **razbijemo** enačbo med **funkcijskima** tipoma

$$\langle \sigma_1 \to \tau_1 = \sigma_2 \to \tau_2, E \mid R \rangle$$

$$\rightsquigarrow$$

$$\langle \sigma_1 = \sigma_2, \tau_1 = \tau_2, E \mid R \rangle$$

Če ima enačba na **eni strani parameter**, jo dodamo* med **rešitve**

$$\langle \alpha = \tau, E \mid R \rangle$$

če α ne nastopa v τ

$$\rightsquigarrow$$

$$\langle E[\tau/\alpha] \mid \alpha \mapsto \tau, R[\tau/\alpha] \rangle$$

V ostalih primerih enačbe **nimajo rešitve**

	α	bool	int	σ'×τ'	σ'→τ'
α	+	+	+	!!	!!
bool	+	+	×	×	×
int	+	×	+	×	×
σ×τ	!!	×	×	⊘	×
σ→τ	!!	×	×	×	⊘

$$\lambda p\, f\, x\, .\, \texttt{if}\ p\ x\ \texttt{then}\ f\ x\ \texttt{else}\ x$$

$$f = \lambda n\,.$$

$$\texttt{if}\ n = 0\ \texttt{then}\ 1$$

$$\texttt{else}\ n * f(n - 1)$$

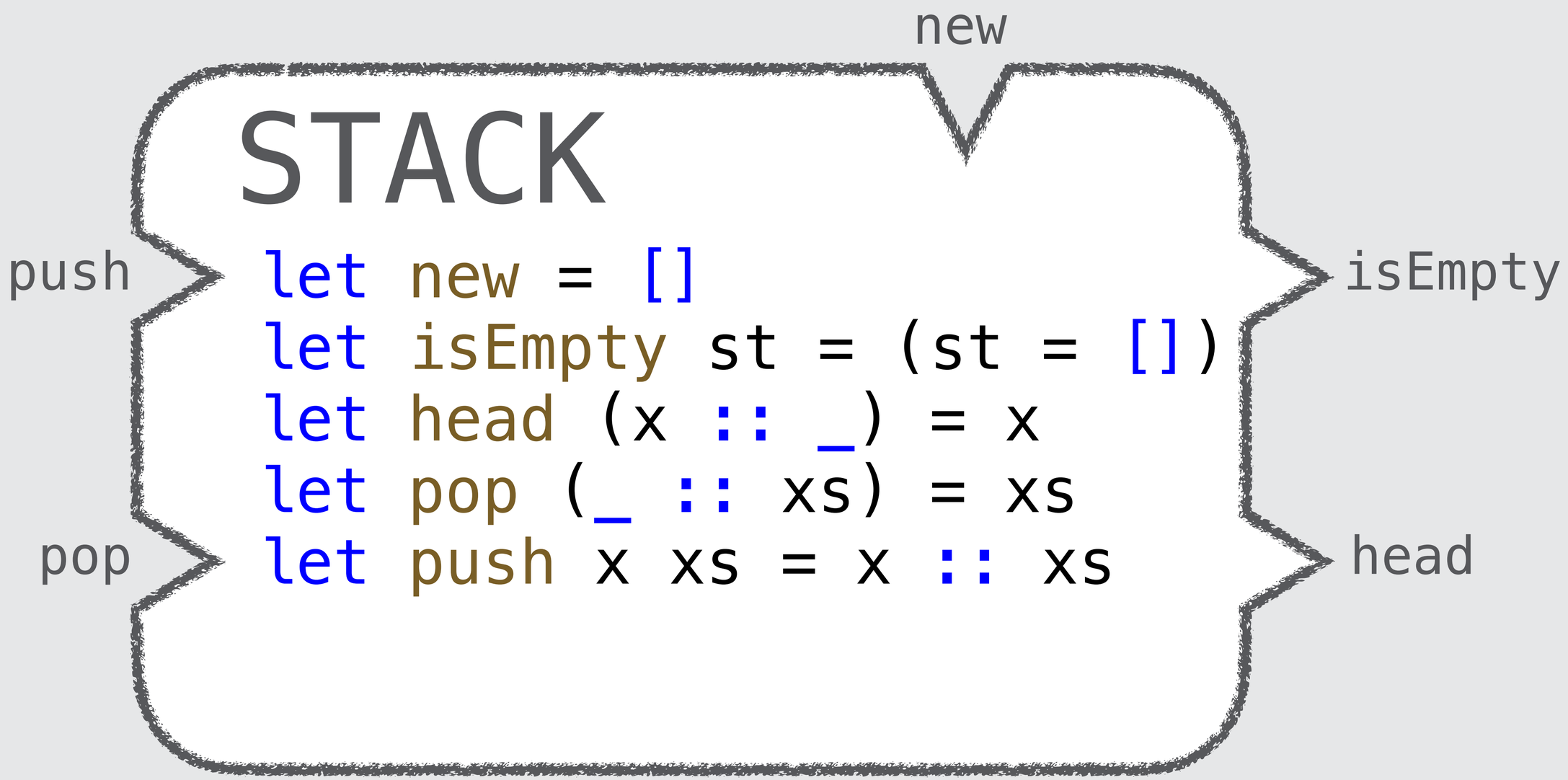

λx . x x

združevanje v jeziku

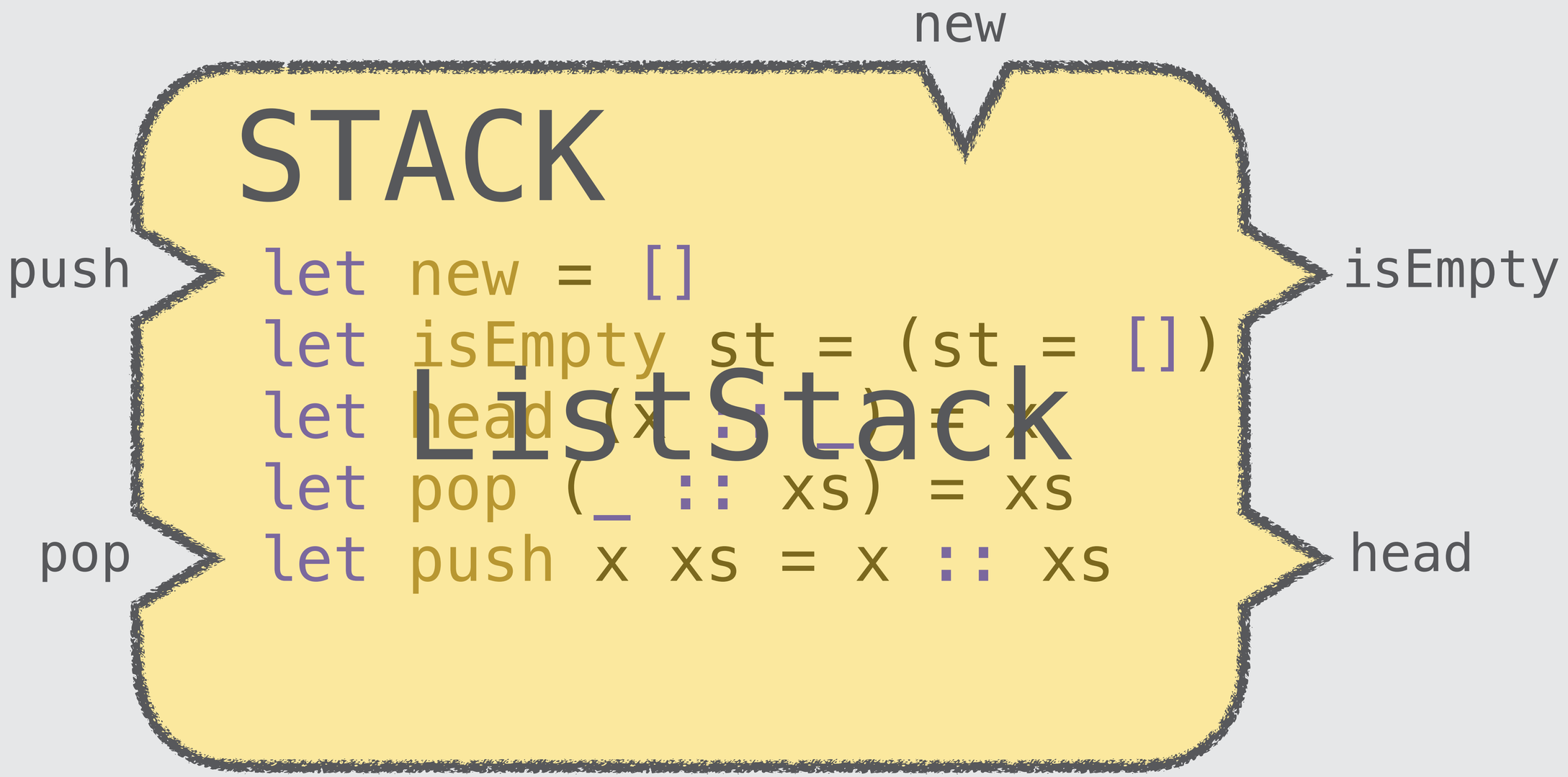
poly

prihodnjič...

Videli bomo, kako podajamo **specifikacije** in **implementacije**



Implementacije bomo skrivali z **abstrakcijo**



Ogledali si bomo **generično programiranje**