

# Aritmetični izrazi

# Programski jeziki

# Programski jeziki so **vmesniki** med **Ijudmi** in **računalniki**



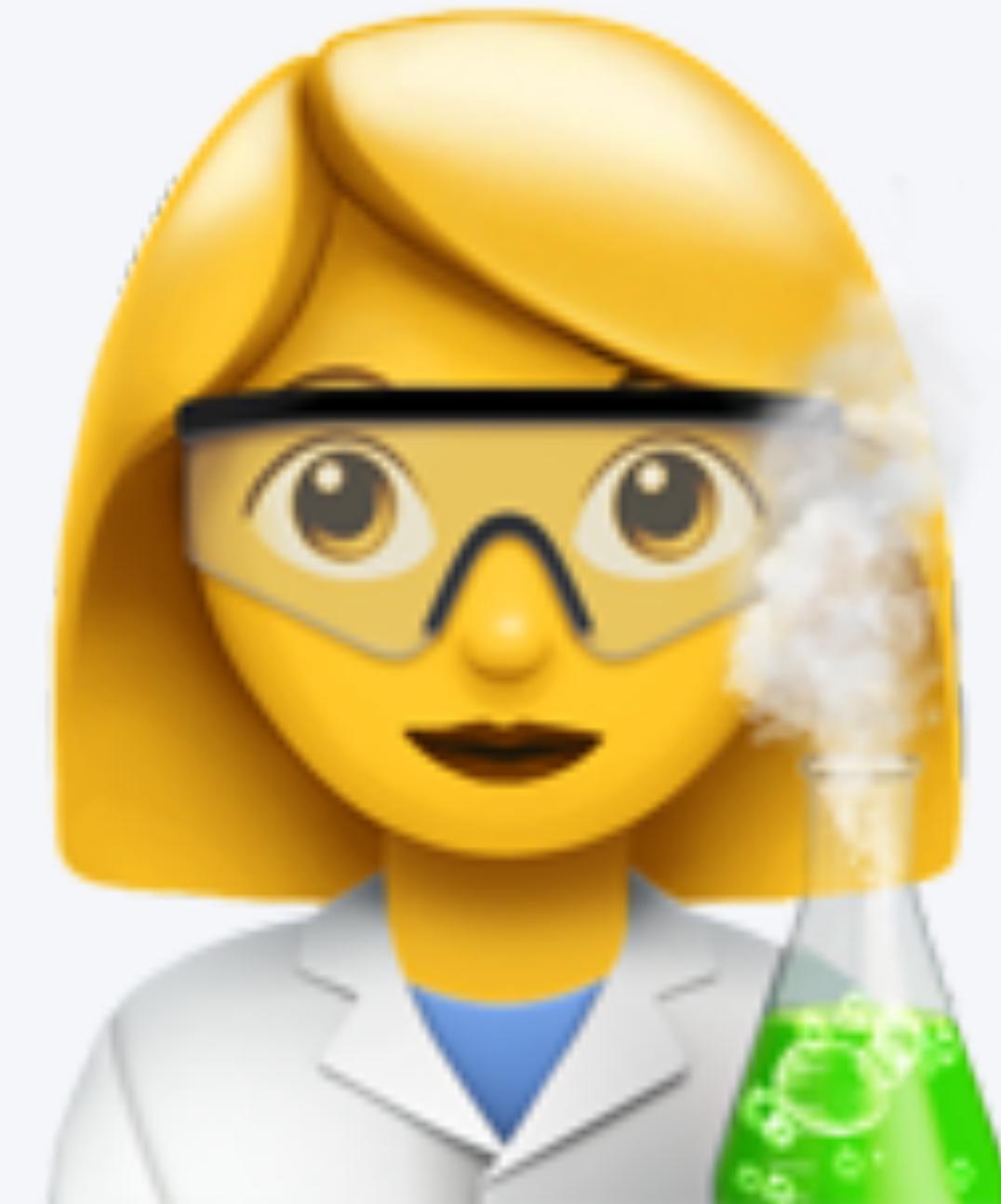
programska  
jezik

A large, hollow arrow pointing from the person emoji towards the computer monitor, containing the text "programska jezik".

Programske jezike uporabljam  
tudi za **medčloveško komunikacijo**



programska  
jezik



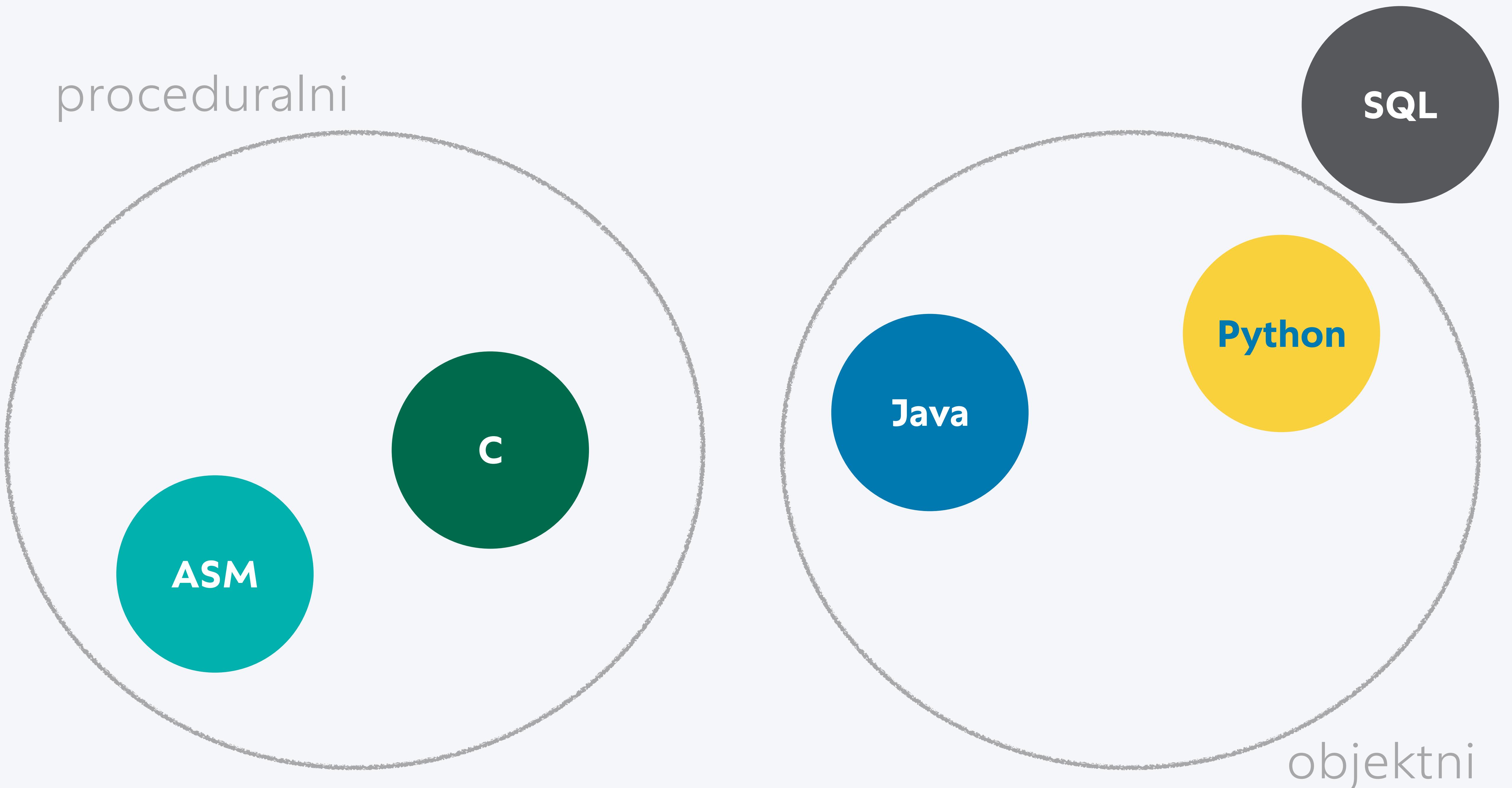
Programske jezike uporabljam  
tudi za **medčloveško komunikacijo**



programska  
jezik



Med študijem ste spoznali  
že **več različnih** programskih jezikov



Namen predmeta PPJ je spoznati **ključna načela  
načrtovanja, implementacije in analize jezikov**

**imperativni**

**deklarativni**

proceduralni

funkcijski

Python

Rust

Java

Haskell

OCaml

Prolog

SQL

ASM

C

objektni

logični

A black and white portrait of Ludwig Wittgenstein in profile, facing right. He has dark hair and is wearing a light-colored, open-collared shirt under a dark, textured jacket.

Meje  
mojega jezika  
so meje  
mojega sveta.

Ludwig Wittgenstein, 1889–1951

Programski jezik je določen s **sintakso** in  
**semantiko**, ki je lahko različnih oblik

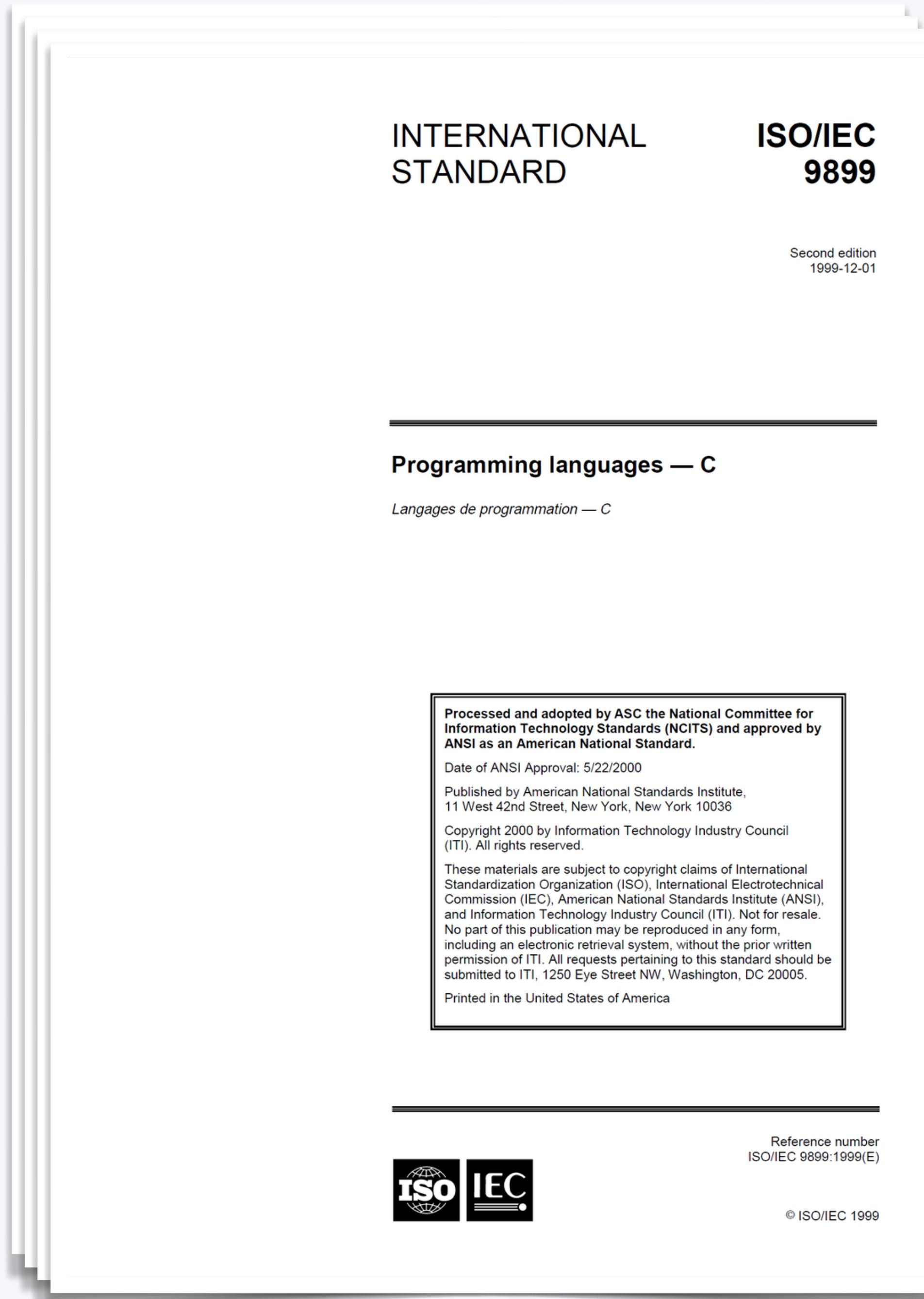
**sintaksa**  
pravila pisanja

**statična**  
**semantika**  
pravila veljavnosti

**denotacijska**  
**semantika**  
matematični pomen

**dinamična**  
**semantika**  
pravila izvajanja

# Sintaksa in semantika jezika sta podana prek specifikacije in/ali implementacije



Formalno spisane specifikacije  
so ponavadi **precej obsežne**

**C23**

~ 742 strani

**C++23**

~ 2124 strani

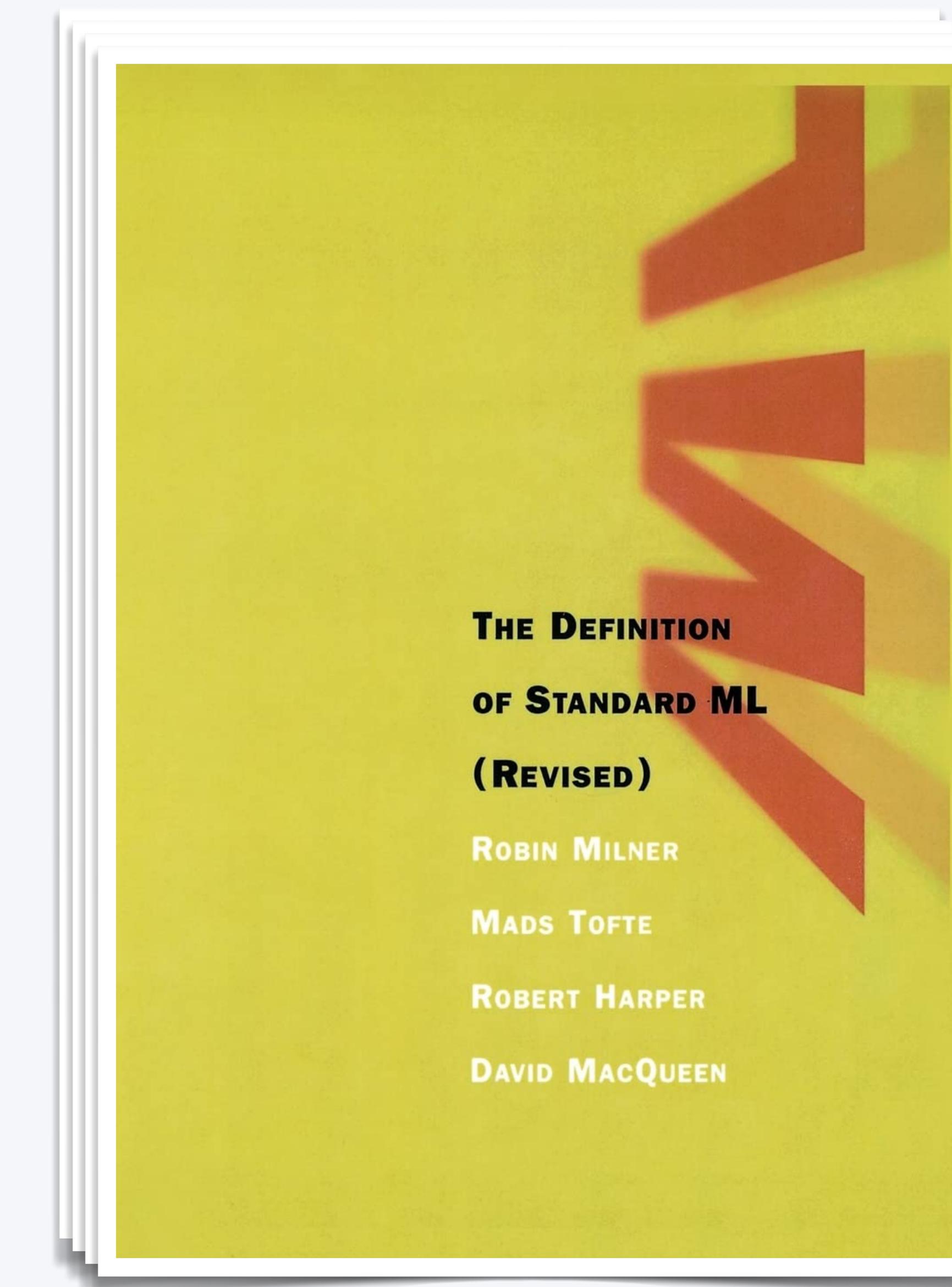
**Java SE 21**

~850 strani

**ECMAScript 2023**

~875 strani

**Standard ML** oz. **SML** je bil eden prvih programskih jezikov z matematično podano specifikacijo



~120 strani

# WebAssembly oz. Wasm je najbolj razširjen programski jezik z matematično podano specifikacijo

**loop** *blocktype instr\** **end**

1. Let  $F$  be the **current frame**.
2. Assert: due to **validation**,  $\text{expand}_F(\text{blocktype})$  is defined.
3. Let  $[t_1^m] \rightarrow [t_2^n]$  be the **function type**  $\text{expand}_F(\text{blocktype})$ .
4. Let  $L$  be the label whose arity is  $m$  and whose continuation is the start of the loop.
5. Assert: due to **validation**, there are at least  $m$  values on the top of the stack.
6. Pop the values  $\text{val}^m$  from the stack.
7. **Enter** the block  $\text{val}^m \text{ instr}^*$  with label  $L$ .

$$F; \text{val}^m \text{ loop } bt \text{ instr}^* \text{ end} \hookrightarrow F; \text{label}_m\{\text{loop } bt \text{ instr}^* \text{ end}\} \text{ val}^m \text{ instr}^* \text{ end} \\ (\text{if } \text{expand}_F(bt) = [t_1^m] \rightarrow [t_2^n])$$

**if** *blocktype instr<sub>1</sub>\** **else** *instr<sub>2</sub>\** **end**

1. Assert: due to **validation**, a value of **value type i32** is on the top of the stack.
2. Pop the value **i32.const c** from the stack.
3. If  $c$  is non-zero, then:
  - a. Execute the block instruction **block blocktype instr<sub>1</sub>\*** **end**.
4. Else:
  - a. Execute the block instruction **block blocktype instr<sub>2</sub>\*** **end**.

$$\begin{aligned} (\text{i32.const } c) \text{ if } bt \text{ instr}_1^* \text{ else } \text{instr}_2^* \text{ end} &\hookrightarrow \text{block } bt \text{ instr}_1^* \text{ end} \\ &\quad (\text{if } c \neq 0) \\ (\text{i32.const } c) \text{ if } bt \text{ instr}_1^* \text{ else } \text{instr}_2^* \text{ end} &\hookrightarrow \text{block } bt \text{ instr}_2^* \text{ end} \\ &\quad (\text{if } c = 0) \end{aligned}$$

**br l**

1. Assert: due to **validation**, the stack contains at least  $l + 1$  labels.
2. Let  $L$  be the  $l$ th label appearing on the stack, starting from the top and counting from zero.

~220 strani

# Implementacija jezika iz programa, ki sledi specifikaciji, izračuna rezultat

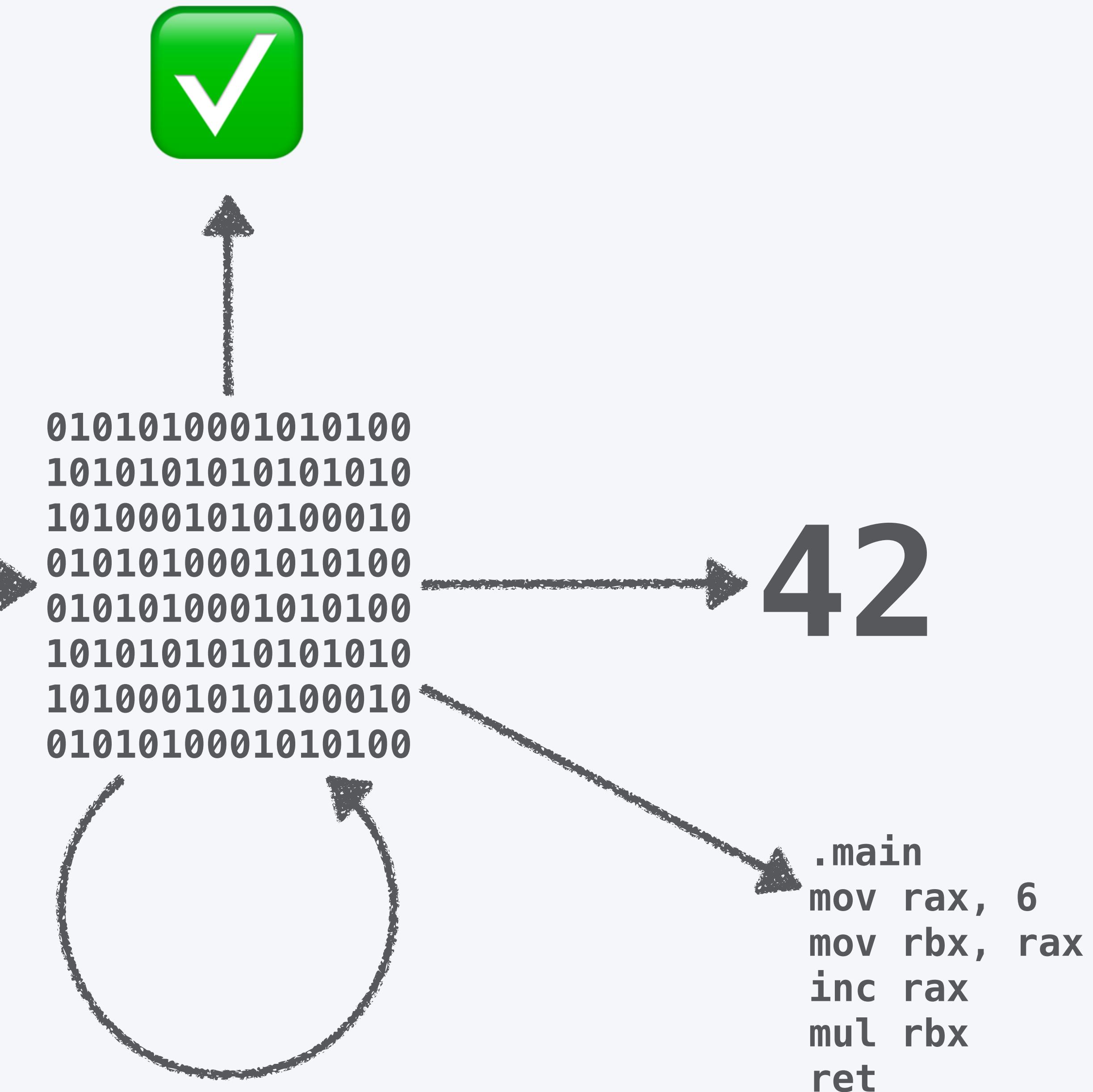
gcc

```
int main()
{
    int x = 6;
    x *= x + 1;
    return x;
}
```

42

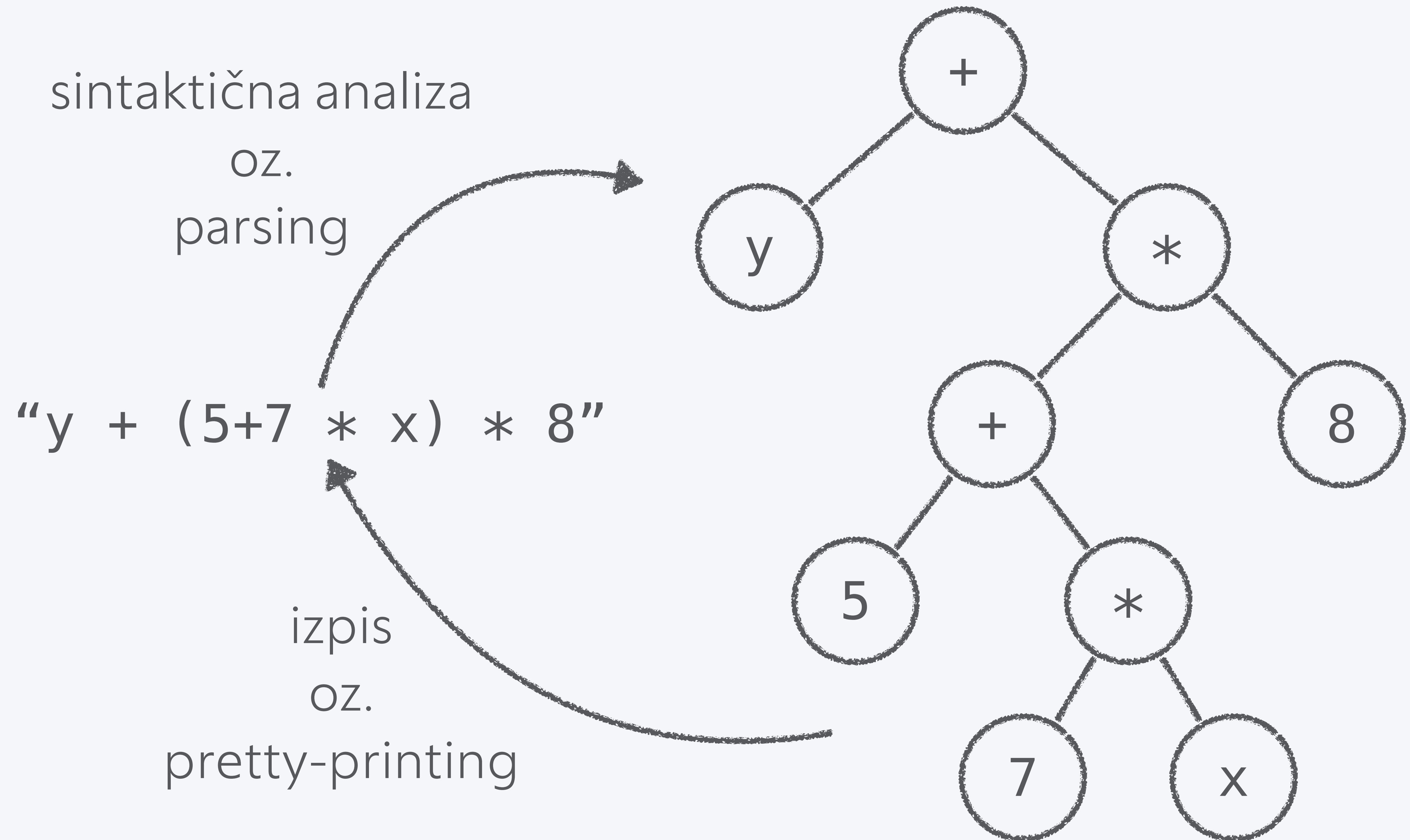
Implementacija je sestavljena  
iz več zaporednih faz

```
int main()
{
    int x = 6;
    x *= x + 1;
    return x;
}
```



# Sintaksa

Sintakso ločimo na **konkretno**, ki je namenjena vnosu,  
in **abstraktno**, ki jo lažje obdelujemo



# Gramatiko konkretne sintakse podamo z **Backus-Naurovo** notacijo oz. **BNF**, sestavljeni iz **produkcijs** z več **alternativami**

```
<izraz> ::= <presledki> <aditivni> <presledki> "\0"
<aditivni> ::= <multiplikativni>
                  | <aditivni> <presledki> "+" <presledki> <multiplikativni>
<multiplikativni> ::= <osnovni>
                  | <multiplikativni> <presledki> "*" <presledki> <osnovni>
<osnovni> ::= <spremenljivka>
                  | <številka>
                  | "(" <presledki> <aditivni> <presledki> ")"
<spremenljivka> ::= <črka> <črke>
<številka> ::= <števka> <števke> | "-" <števka> <števke>
<števka> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<števke> ::= "" | <števka> <števke>
<črka> ::= "a" | "b" | ... | "z"
<črke> ::= "" | <črka> <črke>
<presledek> ::= " " | "\n" | "\t" | "\r"
<presledki> ::= "" | <presledki1>
<presledki1> ::= <presledek> <presledki>
```

John Backus, 1924–2007

Peter Naur, 1928–2016



WikiMedia Commons  
CC-BY-SA Eriktj, Sozi

WikiMedia Commons  
CC-BY-SA Pierre.Lescanne

Pred sintaktično analizo naredimo **leksikalno analizo**, ki niz razbije na zaporedje **leksemov**

```
<izraz> ::= <aditivni> EOF
<aditivni> ::= <multiplikativni>
               | <aditivni> PLUS <multiplikativni>
<multiplikativni> ::= <osnovni>
                      | <multiplikativni> KRAT <osnovni>
<osnovni> ::= <spremenljivka>
              | <številka>
              | OKLEPAJ <aditivni> ZAKLEPAJ
<spremenljivka> ::= SPREMENLJIVKA(string)
<številka> ::= ŠTEVILKA(int)
```

“y + (5+7 \* x) \* 8”

SPREMENLJIVKA(y) PLUS OKLEPAJ ŠTEVILKA(5) PLUS ŠTEVILKA(7)  
KRAT SPREMENLJIVKA(x) ZAKLEPAJ KRAT ŠTEVILKA(8) EOF

# Pri podajanju gramatike moramo paziti na dvoumnost

```
<izraz> ::= <podizraz> EOF
<podizraz> ::= <spremenljivka>
              | <številka>
              | <podizraz> PLUS <podizraz>
              | <podizraz> KRAT <podizraz>
              | OKLEPAJ <podizraz> ZAKLEPAJ
<spremenljivka> ::= SPREMENLJIVKA(string)
<številka> ::= ŠTEVILKA(int)
```

“1 + 2 + 3 \* 4”

# Veliko dvoumnosti lahko razjasnimo, če podamo **prioriteto** in **asociiranost** operatorjev

```
<izraz> ::= <podizraz> EOF
<podizraz> ::= <spremenljivka>
              | <številka>
              | <podizraz> PLUS <podizraz>
              | <podizraz> KRAT <podizraz>
              | OKLEPAJ <podizraz> ZAKLEPAJ
<spremenljivka> ::= SPREMENLJIVKA(string)
<številka> ::= ŠTEVILKA(int)
```

```
%left KRAT
%left PLUS
```

“1 + 2 + 3 \* 4”

sintaksa jezika  
cać

# Sintaksi se boste bolj posvetili pri predmetu **Prevajalniki**

Ali bolj tehnično: Predstavitev zgradbe, delovanja in izdelave prevajalnika za prevajanje programskih jezikov v zbirnik; posamezna poglavja ustrezajo posameznim fazam prevajalnika: leksikalna analiza, sintaksna analiza, abstraktna sintaksa, semantična analiza, klicni zapisi, vmesna koda, osnovni bloki, izbira strojnih ukazov, analiza aktivnosti spremenljivk, dodeljevanje registrov.

## ■ ŠTUDIJSKI PROGRAMI

- Računalništvo in informatika
- Računalništvo in matematika

## ■ PORAZDELITEV UR NA SEMESTER

45  
ur

predavanj

30  
ur

laboratorijskih vaj

## ■ IZVAJALCI



**doc. dr. Boštjan Slivnik**

nosilec predmeta

Prostor: R2.14 - Kabinet

[PODROBNOSTI](#)

Mi bomo BNF uporabljali predvsem za  
podajanje **abstraktne sintakse**

izraz  $e ::= n \mid x \mid e_1 + e_2 \mid e_1 * e_2$



[imgflip.com](http://imgflip.com)

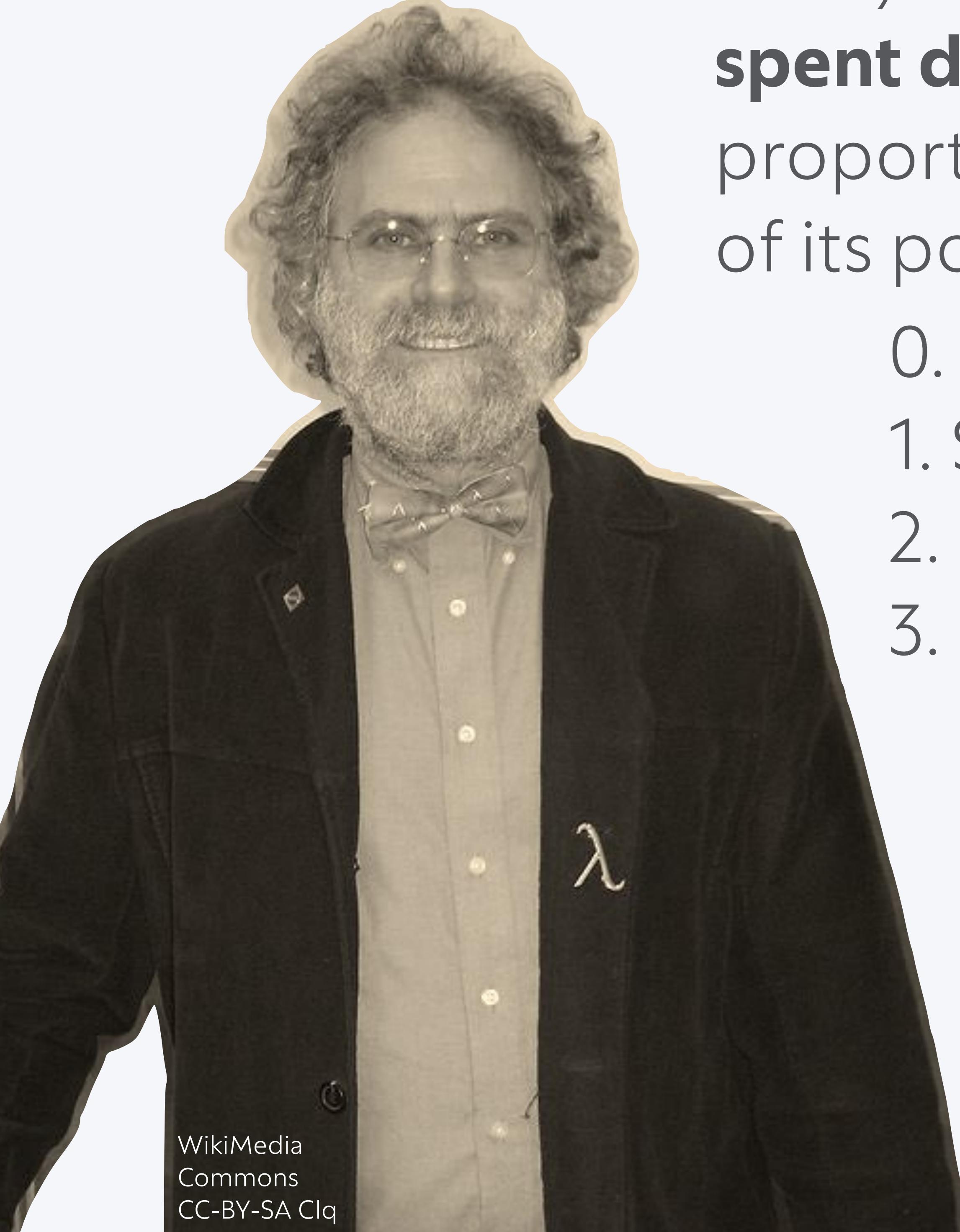


**BRACE YOURSELVES**



'IS COMING



A black and white portrait of Philip Wadler, a man with curly hair and a beard, wearing glasses, a bow tie, and a dark jacket over a light shirt. A small Greek letter λ is visible on his left lapel.

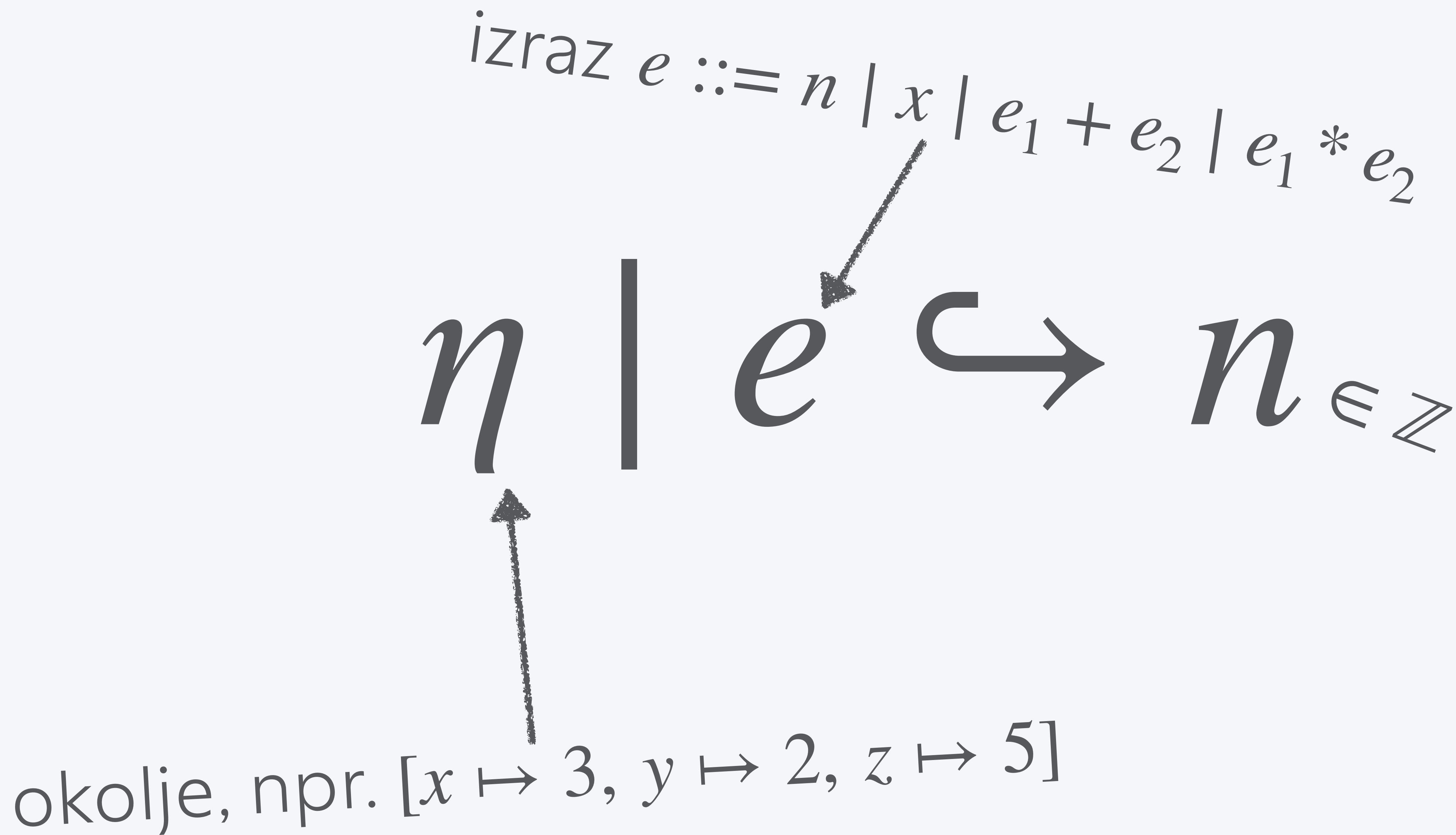
In any language design, the total time  
**spent discussing** a feature in this list is  
proportional to **two raised to the power**  
of its position.

0. Semantics
1. Syntax
2. Lexical syntax
3. Lexical syntax of comments

Philip Wadler, 1956–

# Operacijska semantika

Operacijsko semantiko bomo podali  
prek **relacije velikih korakov**



Definicije relacij podajamo prek **pravil sklepanja**,  
ki imajo nad črto **predpostavke**, pod njo pa **sklep**

$$\frac{P_1 \quad P_2 \quad \dots \quad P_n}{S}$$

Nekaj pravil sklepanja,  
ki določajo **sorodstvo**

---

$x$  prednik  $x$

$x$  prednik  $y$      $y$  stars  $z$

---

$x$  prednik  $z$

---

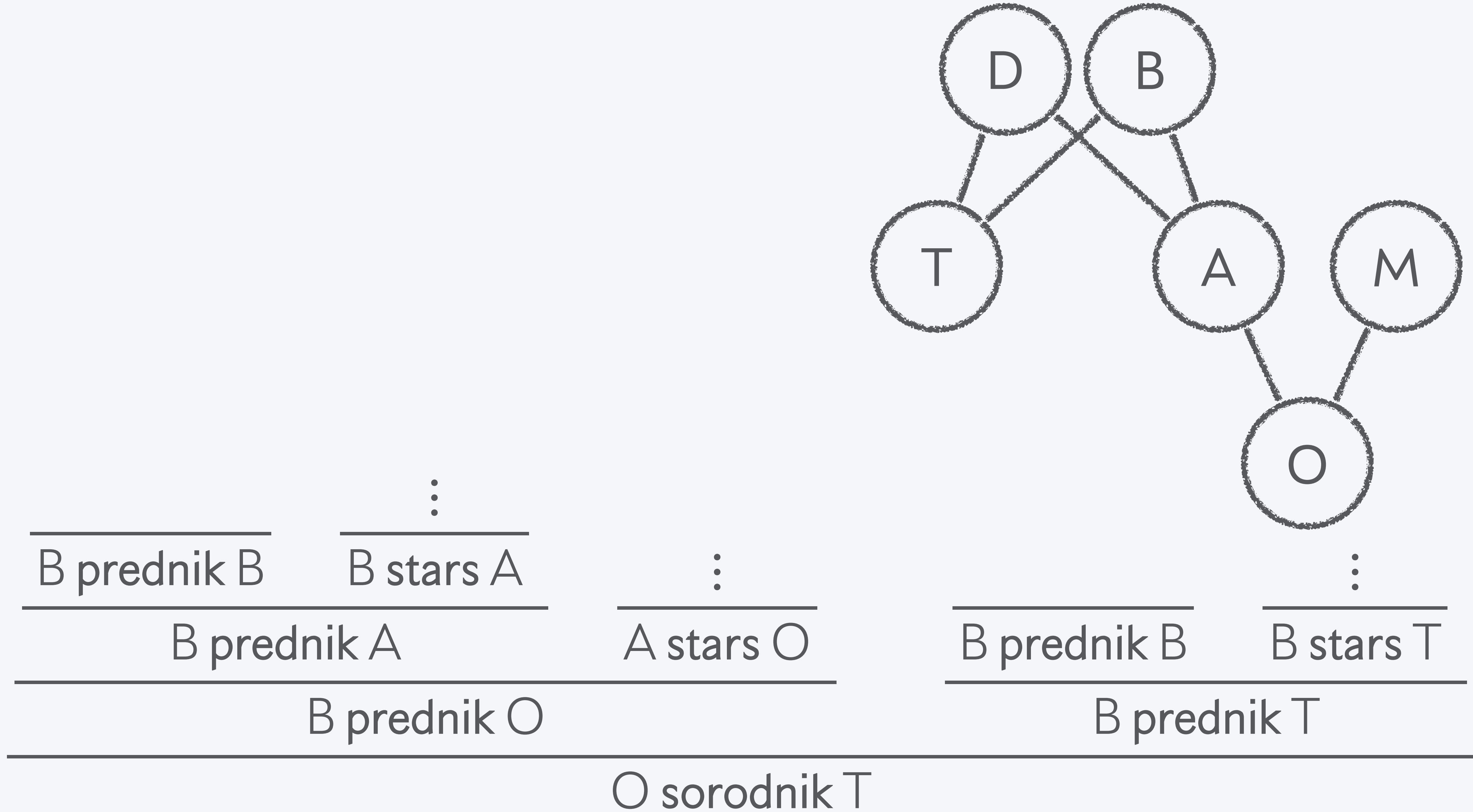
$x$  prednik  $y$

$x$  prednik  $z$

---

$y$  sorodnik  $z$

Ob danih pravilih sklepanja vsak veljaven **sklep**  
utemeljimo z **drevesom izpeljave**



# Pravila sklepanja, ki **induktivno** podajajo množico naravnih števil $\mathbb{N}$

$$\underline{0 \in \mathbb{N}}$$

$$\frac{n \in \mathbb{N}}{\underline{n^+ \in \mathbb{N}}}$$

# Pravila sklepanja, ki **induktivno** podajajo semantiko **velikih korakov**

$$\eta \mid e \hookrightarrow n$$

$$\frac{\eta(x) = n}{\eta \mid x \hookrightarrow n}$$

$$\eta \mid n \hookrightarrow n$$

$$\frac{\eta \mid e_1 \hookrightarrow n_1 \quad \eta \mid e_2 \hookrightarrow n_2 \quad n_1 + n_2 = n}{\eta \mid e_1 + e_2 \hookrightarrow n}$$

$$\frac{\eta \mid e_1 \hookrightarrow n_1 \quad \eta \mid e_2 \hookrightarrow n_2 \quad n_1 \cdot n_2 = n}{\eta \mid e_1 * e_2 \hookrightarrow n}$$

semantika jezika  
cač

Operacijsko semantiko lahko podamo  
tudi prek **relacije malih korakov**

$$\eta \mid e \hookrightarrow n$$

$$\eta \mid e_1 \mapsto e_2$$

# Pravila sklepanja, ki induktivno podajajo semantiko **malih korakov**

$$\eta \mid e_1 \mapsto e_2$$

$$\frac{\eta(x) = n}{\eta \mid x \mapsto n}$$

$$\frac{\eta \mid e_1 \mapsto e'_1}{\eta \mid e_1 + e_2 \mapsto e'_1 + e_2}$$

$$\frac{\eta \mid e_2 \mapsto e'_2}{\eta \mid n_1 + e_2 \mapsto n_1 + e'_2}$$

$$\frac{n_1 + n_2 = n}{\eta \mid n_1 + n_2 \mapsto n}$$

$$\frac{\eta \mid e_1 \mapsto e'_1}{\eta \mid e_1 * e_2 \mapsto e'_1 * e_2}$$

$$\frac{\eta \mid e_2 \mapsto e'_2}{\eta \mid n_1 * e_2 \mapsto n_1 * e'_2}$$

$$\frac{n_1 \cdot n_2 = n}{\eta \mid n_1 * n_2 \mapsto n}$$

prihodnjič...

# Spoznali bomo Turingovo poln imperativni jezik **IMP**

aritmetični izraz  $e ::= x \mid n \mid e_1 + e_2 \mid e_1 * e_2$

logični izraz  $b ::= \text{true} \mid \text{false} \mid e_1 = e_2 \mid e_1 < e_2 \mid b_1 \text{ and } b_2 \mid b_1 \text{ or } b_2 \mid \text{not } b$

ukaz  $c ::= \text{skip} \mid x := e \mid c_1; c_2 \mid$

$\text{while } b \text{ do } c \text{ done} \mid$

$\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}$

Programom bomo podali **denotacijsko semantiko**,  
torej jih interpretirali z **matematičnimi objekti**

$\llbracket \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end} \rrbracket(\eta)$

$$= \begin{cases} \llbracket c_1 \rrbracket(\eta) & \llbracket b \rrbracket(\eta) = \top \\ \llbracket c_2 \rrbracket(\eta) & \llbracket b \rrbracket(\eta) = \perp \end{cases}$$

Razmislili bomo, kdaj sta  
dva programa **ekvivalentna**

if  $b$  then  $c_1$  else  $c_2$  end

$\approx$

if (not  $b$ ) then  $c_2$  else  $c_1$  end