

# Programiranje z omejitvami

prejšnjič...

# Logično programiranje je nov pristop k programiranju

## proceduralno programiranje

Iskani rezultat dosežemo z **zaporedjem ukazov**.

## funkcijsko programiranje

Iskani rezultat opišemo s **sestavljanjem izrazov**.

## logično programiranje

Iskani rezultat določimo z **logičnimi formulami**.

# Spoznali smo programski jezik Prolog

$\forall x . \text{produkt}(x, 0, 0)$

$\forall x, y, z, w . \text{produkt}(x, y, w) \wedge \text{vsota}(x, w, z)$

$\Rightarrow \text{produkt}(x, y^+, z)$

?  $\exists p, q . \text{produkt}(p, q, 4)$



```
prod( _, zero, zero ).  
prod(X, succ(Y), Z) :-  
    prod(X, Y, W),  
    sum(X, W, Z).
```

```
?- prod(P, Q, succ(succ(succ(succ(zero))))).
```



# Splošen postopek reševanja $\exists x_1, \dots, x_m . P(t_1, \dots, t_n)$

## iskanje

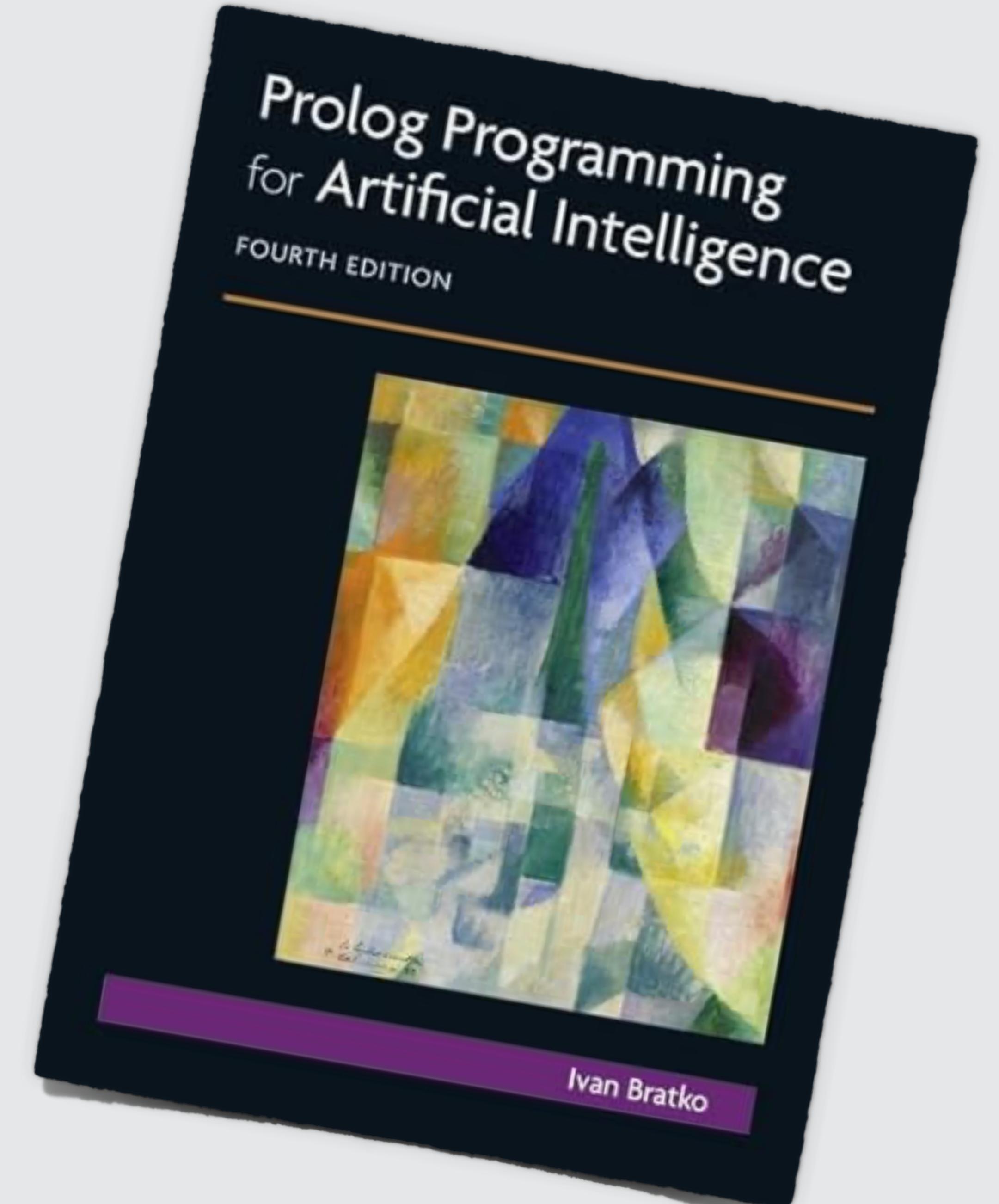
**Zaporedoma** gremo čez formule in izberemo **prvo** formulo oblike  
 $\forall y_1, \dots, y_k . \varphi_1 \wedge \dots \wedge \varphi_\ell \Rightarrow P(u_1, \dots, u_n).$

## združevanje

Določimo **najbolj splošne** vrednosti  $\sigma$  spremenljivk  $x_1, \dots, x_k$  in  $y_1, \dots, y_m$ ,  
da velja  $\sigma(t_1) = \sigma(u_1), \dots, \sigma(t_n) = \sigma(u_n)$ .

## sestopanje

**Rekurzivno rešimo**  $\sigma(\varphi_1), \dots, \sigma(\varphi_\ell)$ , kjer spremenljivke **kvantificiramo eksistenčno**.  
Če kakšna predpostavka nima rešitve, **poишčemo naslednjo** združljivo formulo  $H'$ , **če obstaja**.



Ivan Bratko, 1946-



$\forall n . \text{vsota}(n, 0, n)$

$\forall k, m, n . \text{vsota}(k, m, n)$

$\Rightarrow \text{vsota}(k, m^+, n^+)$

?     $\exists n . \text{vsota}(0^+, n, 0^{++})$

$$\forall x . \text{sodo}(x) \Rightarrow \text{liho}(x^+)$$
$$\forall y . \text{liho}(y) \Rightarrow \text{sodo}(y^+)$$
$$\text{sodo}(0)$$

?       $\exists z . \text{liho}(z)$

# S Hornovimi formulami izrazimo induktivno podane relacije

$$(\eta, c) \mapsto (\eta', c')$$

$$\frac{\eta \mid e \hookrightarrow n}{(\eta, x := e) \mapsto (\eta[x \mapsto n], \text{skip})}$$

$$\frac{(\eta, c_1) \mapsto (\eta', c'_1)}{(\eta, c_1; c_2) \mapsto (\eta', c'_1; c_2)}$$

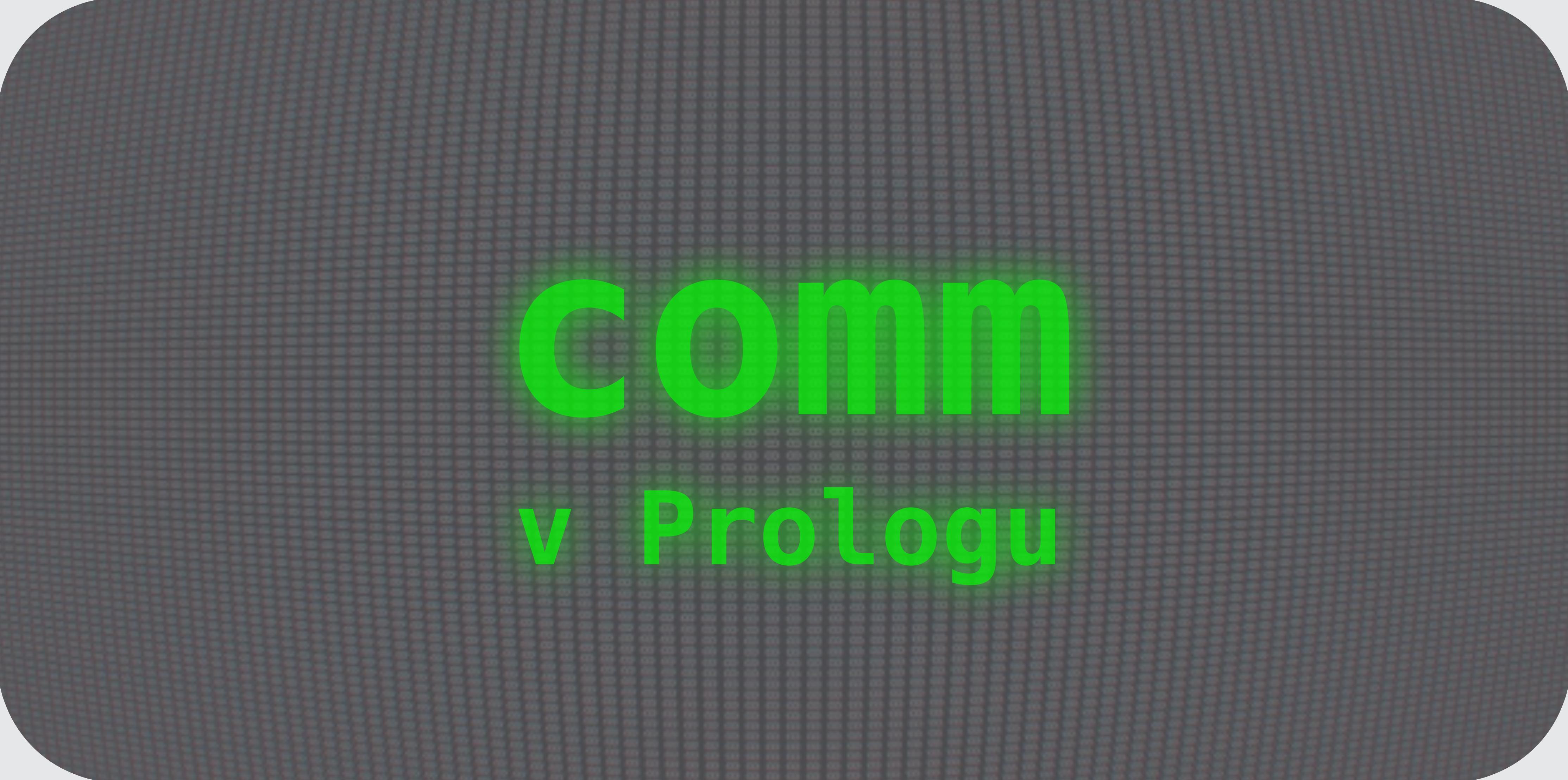
$$(\eta, \text{skip}; c_2) \mapsto (\eta, c_2)$$

$$\frac{\eta \mid b \hookrightarrow \text{true}}{(\eta, \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}) \mapsto (\eta, c_1)}$$

$$\frac{\eta \mid b \hookrightarrow \text{false}}{(\eta, \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}) \mapsto (\eta, c_2)}$$

$$\frac{\eta \mid b \hookrightarrow \text{true}}{(\eta, \text{while } b \text{ do } c \text{ done}) \mapsto (\eta, c; \text{while } b \text{ do } c \text{ done})}$$

$$\frac{\eta \mid b \hookrightarrow \text{false}}{(\eta, \text{while } b \text{ do } c \text{ done}) \mapsto (\eta, \text{skip})}$$



comm  
v Prologu

# Aritmetika v Prologu

# Prolog podpira več vrst **enakosti**

## 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
```

**učinkovita  
fakulteta  
v Prologu**

# Omejitve na celih številih

# Knjižnica **CLP(FD)** podpira aritmetične omejitve

```
?- X + 2 #= 3.
```

```
X = 1.
```

```
?- X * 5 #= 210.
```

```
X = 42.
```

```
?- 3 #< X, 4 * X #< 25.
```

```
X in 4..6.
```

**učinkovita  
fakulteta  
z omejitvami**

Programiranje z omejitvami je omogočeno **na več domenah**

**CLP(FD)**

Končne domene & **cela števila**  $\mathbb{Z}$

**CLP(QR)**

**Racionalna** števila  $\mathbb{Q}$  & **realna** števila  $\mathbb{R}$

**CLP(B)**

**Booleova** algebra

Na domeni celih števil poznamo **intervalske omejitve**

```
?- X in 1..10, X in 5..13.  
X in 5..10.
```

```
?- [X,Y] ins 1..20, X #< Y.  
X in 1..19,  
X #=< Y + -1,  
Y in 2..20.
```

Če želimo, lahko naštejemo vse rešitve **danih omejitv**

```
?- 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 .
```

**pitagorejske  
trojice**

# Poznamo 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].
```



sudoku

**prihodnjič...**

Pogledali bomo, kdaj je nek tip **podtip** drugega

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

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

# Za primer si bomo ogledali objektno programiranje

```
object (self)
    val mutable x = 10.0
    val mutable y = 7.5

    method get_x = x
    method get_y = y
    method premakni dx dy =
        x <- self#get_x +. dx;
        y <- self#get_x +. dy
end
```