# IB107 Vyčíslitelnost a složitost

## standardní numerace, problém zastavení, věta o numeraci

Jan Strejček

Fakulta informatiky
Masarykova univerzita

# numerace množiny

## Definice 5.1 (numerace množiny)

*Numerace množiny* $M$ *je surjektivní funkce* $\nu : \mathbb{N} \to M$.
*Je-li* $\nu$ *navíc totální, mluvíme o totální numeraci množiny.*

příklady

- numerace množiny $M = \{a, b, c\}$

- totální numerace množiny $M = \{a, b, c\}$

# numerace množiny

- numerace množiny $\{0, 2, 4, 6, 8, \ldots\}$
- numerace množin $\mathbb{Z}$ a $\mathbb{Q}$
- numerovat lze libovolnou nejvýše spočetnou množinu
- chceme definovat totální numeraci $j$-árních vyčíslitelných funkcí $\mathcal{P}^{(j)}$ (pro každé $j \geq 1$)

# index programu

- každému programu přiřadíme číslo z $\mathbb{N}$
- zvolíme techniku, kterou použil Kurt Gödel
- předpokládáme, že programy používají pouze proměnné $x_1, x_2, \ldots$
- definujeme funkci *code* : $\mathbb{P} \to \mathbb{N}$ pro všechna $i, j, m \geq 1$ a libovolné příkazy $\delta, \delta_1, \delta_2, \ldots$

$$
\begin{aligned}
\mathit{code}(x_i := 0) &= 2^i \\
\mathit{code}(x_i := x_j - 1) &= 3^i 5^j \\
\mathit{code}(x_i := x_j + 1) &= 7^i 11^j \\
\mathit{code}(\underline{\textbf{while}}\ x_i \neq x_j\ \underline{\textbf{do}}\ \delta) &= 13^i 17^j 19^{\mathit{code}(\delta)} \\
\mathit{code}(\underline{\textbf{begin}}\ \delta_1; \delta_2; \ldots \delta_m\ \underline{\textbf{end}}) &= 23^{\mathit{code}(\delta_1)} 29^{\mathit{code}(\delta_2)} \ldots (p_{8+m})^{\mathit{code}(\delta_m)} \\
\mathit{code}(\underline{\textbf{begin}}\ \ \underline{\textbf{end}}) &= 1
\end{aligned}
$$

kde $p_i$ je $i$-té prvočíslo, přičemž první prvočíslo je 2

- *code*($P$) nazýváme index programu $P$

# příklad

$$
\begin{aligned}
code(x_i := 0) &= 2^i \\
code(x_i := x_j - 1) &= 3^i 5^j \\
code(x_i := x_j + 1) &= 7^i 11^j \\
code(\underline{\textbf{while}}\ x_i \neq x_j\ \underline{\textbf{do}}\ \delta) &= 13^i 17^j 19^{code(\delta)} \\
code(\underline{\textbf{begin}}\ \delta_1; \delta_2; \ldots \delta_m\ \underline{\textbf{end}}) &= 23^{code(\delta_1)} 29^{code(\delta_2)} \ldots (p_{8+m})^{code(\delta_m)} \\
code(\underline{\textbf{begin}}\ \ \underline{\textbf{end}}) &= 1
\end{aligned}
$$

$P_{empty}$: $code(P_{empty}) =$

**begin**
  $x_1 := 0;$
  $x_2 := x_1 + 1;$
  **while** $x_1 \neq x_2$ **do begin  end**
**end**

# vlastnosti funkce *code*

## Lemma 5.3

*Funkce code* : $\mathbb{P} \to \mathbb{N}$ *je injektivní a totální.*

**Důkaz:**

■

## Tvrzení

*Funkce code* : $\mathbb{P} \to \mathbb{N}$ *není surjektivní.*

**Důkaz:** např. číslo 12 není indexem žádného programu. ■

# standardní numerace programů

$P_{empty}$: **begin**
$x_1 := 0;$
$x_2 := x_1 + 1;$
**while** $x_1 \neq x_2$ **do begin end**
**end**

## Definice 5.4 (standardní numerace programů)

*Standardní (kanonická) numerace programů je funkce* $num : \mathbb{N} \mapsto \mathbb{P}$ *definovaná takto:*

$$num(n) = \begin{cases} code^{-1}(n) & \text{jestliže } n \in range(code) \\ P_{empty} & \text{jinak} \end{cases}$$

Namísto $num(0), num(1), \ldots$ obvykle píšeme $P_0, P_1, \ldots$.

# standardní numerace vyčíslitelných funkcí

## Definice 5.5 (standardní numerace vyčíslitelných funkcí)

*Nechť $j \geq 1$. Standardní (kanonická) numerace $j$-árních vyčíslitelných funkcí je funkce $\varphi^{(j)} : \mathbb{N} \mapsto \mathcal{P}^{(j)}$ definovaná takto:*

$$\varphi^{(j)}(i) = \varphi^{(j)}_{num(i)}$$

*Index vyčíslitelné funkce $f \in \mathcal{P}^{(j)}$ je číslo $i$ splňující $f = \varphi^{(j)}(i)$.*

Místo $\varphi^{(j)}(0), \varphi^{(j)}(1), \ldots$ obvykle píšeme $\varphi^{(j)}_0, \varphi^{(j)}_1, \ldots$.
Pro $j = 1$ píšeme jen $\varphi_0, \varphi_1, \ldots$.

Vyčíslitelných $j$-árních funkcí je pouze spočetně mnoho.

Všech vyčíslitelných funkcí je pouze spočetně mnoho.

# počet indexů funkce

**Lemma 5.6**

*Každá vyčíslitelná funkce má nekonečně mnoho různých indexů.*

**Důkaz:**

■

# problém zastavení

- existují funkce, které nejsou vyčíslitelné
- neexistuje algoritmus rozhodující, zda daný program na svém indexu zastaví

**Věta 5.7**

*Funkce* $f : \mathbb{N} \mapsto \mathbb{N}$ *definovaná vztahem*

$$f(i) = \begin{cases} 1 & \textit{jestliže } \varphi_i(i) \textit{ je definováno} \\ 0 & \textit{jestliže } \varphi_i(i) \textit{ není definováno} \end{cases}$$

*není vyčíslitelná.*

# důkaz nerozhodnutelnosti problému zastavení

$$f(i) = \begin{cases} 1 & \text{jestliže } \varphi_i(i) \text{ je definováno} \\ 0 & \text{jestliže } \varphi_i(i) \text{ není definováno} \end{cases}$$

**Důkaz:** **(sporem)** Nechť $f$ je vyčíslitelná funkce. Pak program *confuse*

```
begin
  while f(x1) = 1 do begin  end;
  x1 := 1
end
```

počítá funkci

$$\psi(i) = \begin{cases} \perp & \text{jestliže } f(i) = 1 \\ 1 & \text{jestliže } f(i) = 0 \end{cases}$$

Nechť $e$ je index programu *confuse*. Pak

$$\psi(e) = \varphi_e(e) = \begin{cases} \perp \\ 1 \end{cases}$$

# diagonalizační metoda

| | 0 | 1 | 2 | … | $n$ | … |
|---|---|---|---|---|---|---|
| $P_0$ | ↑ | ↓ | ↓ | … | ↑ | … |
| $P_1$ | ↑ | ↓ | ↑ | … | ↓ | … |
| $P_2$ | ↓ | ↓ | ↓ | … | ↓ | … |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋱ | ⋮ | |
| $P_n$ | ↓ | ↓ | ↑ | … | ↑ | … |
| ⋮ | ⋮ | ⋮ | ⋮ | | ⋮ | ⋱ |
| *confuse* | ↓ | ↑ | ↑ | … | ↓ | … |

# věta o numeraci

## Věta 5.10 (věta o numeraci)

*Pro každé $j \geq 1$ existuje vyčíslitelná funkce $\Phi : \mathbb{N}^{j+1} \to \mathbb{N}$ taková, že pro všechna $e, a_1, \ldots, a_j \in \mathbb{N}$ platí*

$$\Phi(e, a_1, \ldots, a_j) = \varphi_e^{(j)}(a_1, \ldots, a_j).$$

- $\Phi$ nazýváme univerzální funkce pro standardní numeraci $j$-árních vyčíslitelných funkcí
- numerace s vyčíslitelnou univerzální funkcí někdy nazýváme efektivní numerace
- větě se také říká utm-věta (universal Turing machine)

# poznámky k důkazu věty o numeraci

intuitivní implementace univerzální funkce $\Phi$

1. ze vstupu $(e, a_1, \ldots, a_j)$ dekóduj program $P_e$
2. simuluj program $P_e$ na vstupu $(a_1, \ldots, a_j)$
3. pokud simulovaný program $P_e$ skončí, dej na výstup jeho výslednou hodnotu

úskalí intuitivní implementace

- simulovný program může obsahovat libovolně velký (konečný) počet proměnných
- implementace $\Phi$ má ale fixní počet proměnných

$\rightarrow$ kódování více proměnných do jedné

# párující funkce

## Definice 5.12 (párující funkce)

*Totálně vyčíslitelné bijekci* $f : \mathbb{N}^2 \mapsto \mathbb{N}$ *říkáme párující funkce.*

$$
\begin{array}{cccc}
(0,0) & (0,1) & (0,2) & (0,3) & \dots \\
(1,0) & (1,1) & (1,2) & (1,3) & \dots \\
(2,0) & (2,1) & (2,2) & (2,3) & \dots \\
(3,0) & (3,1) & (3,2) & (3,3) & \dots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

# párující funkce

$$\begin{array}{cccc} (0,0) & (0,1) & (0,2) & (0,3) & \dots \\ (1,0) & (1,1) & (1,2) & (1,3) & \dots \\ (2,0) & (2,1) & (2,2) & (2,3) & \dots \\ (3,0) & (3,1) & (3,2) & (3,3) & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

## Lemma 5.13

*Funkce* $\tau : \mathbb{N}^2 \mapsto \mathbb{N}$ *definovaná vztahem*

$$\tau(i,j) = \frac{(i+j)(i+j+1)}{2} + i$$

*je párující funkce.*

**Důkaz:** $\tau$ je bijekce (viz konstrukce) a totálně vyčíslitelná. ■

# projekce

## Lemma 5.14

*Funkce* $\pi_1 : \mathbb{N} \mapsto \mathbb{N}$ *a* $\pi_2 : \mathbb{N} \mapsto \mathbb{N}$ *definované jako*

$$\pi_1(\tau(i,j)) = i \quad a \quad \pi_2(\tau(i,j)) = j$$

*jsou totálně vyčíslitelné funkce. Funkce nazýváme projekce.*

**Důkaz:** program pro $\pi_1$

**begin**
  $x_2 := 0;\ x_3 := 0;$
  **while** $\tau(x_2, 0) < x_1$ **do** $x_2 := x_2 + 1;$
  **while** $\tau(x_2, x_3) \neq x_1$ **do begin** $x_2 := x_2 - 1;\ x_3 := x_3 + 1$ **end**;
  $x_1 := x_2$
**end**

■

# standardní kódovací funkce

## Definice 5.15 (standardní kódovací funkce)

*Pro všechna* $k \geq 1$ *definujeme funkce* $\tau_k : \mathbb{N}^k \mapsto \mathbb{N}$ *následně:*

$$\begin{aligned} \tau_1(i_1) &= i_1 \\ \tau_k(i_1, \ldots, i_k) &= \tau(\tau_{k-1}(i_1, \ldots, i_{k-1}), i_k) \qquad \text{pro } k > 1 \end{aligned}$$

*Tyto funkce nazýváme standardní kódovací funkce. Odpovídající projekce* $\pi_{kl} : \mathbb{N} \mapsto \mathbb{N}$ *jsou pro všechna* $1 \leq l \leq k$ *definovány takto:*

$$\begin{aligned} \pi_{k1}(\tau_k(i_1, \ldots, i_k)) &= i_1 \\ &\vdots \\ \pi_{kk}(\tau_k(i_1, \ldots, i_k)) &= i_k \end{aligned}$$

# počítání projekcí

## Lemma 5.16

*Existuje pevné číslo $n$ takové, že pro každé $k, l \in \mathbb{N}$ splňující $1 \leq l \leq k$ je funkce $\pi_{kl}$ vyčíslitelná programem, který používá nejvýše $n$ proměnných.*

**Důkaz:**

- nechť $m$ je počet proměnných potřebný k výpočtu $\tau$
- programy počítající $\pi_1, \pi_2$ vystačí s $m + 3$ proměnnými
- $\pi_{kl}$ lze počítat pomocí $\pi_1, \pi_2$

$$\begin{aligned} \pi_{k1}(i) &= \pi_1^{k-1}(i) \\ \pi_{kl}(i) &= \pi_2(\pi_1^{k-l}(i)) \quad \text{pro } l > 1 \end{aligned}$$

# počítání funkcí s omezeným počtem proměnných

## Lemma 5.18

*Ke každému $j \geq 1$ existuje číslo $r$ a totálně vyčíslitelná funkce* *short* $: \mathbb{N} \mapsto \mathbb{N}$ *taková, že*

- $\varphi_e^{(j)} = \varphi_{short(e)}^{(j)}$
- *a program $P_{short(e)}$ používá nejvýše $j + r$ proměnných*

**Důkaz:** Nechť program $P_e$ používá proměnné $x_1, \ldots, x_k$.
Program $P_{short(e)}$

- zakóduje tyto proměné do proměnné $U$
- simuluje program $P_e$ s využitím projekcí
- používá pouze
  - $j$ vstupních proměnných
  - $m$ proměnných pro výpočet $\tau$
  - $n$ proměnných pro výpočet projekcí (viz předchozí lemma)
  - pomocné proměnné $U, V, W$

# počítání funkcí s omezeným počtem proměnných

program $P_{short(e)}$:

**begin**

$U := \tau(x_1, x_2);$

$U := \tau(U, x_3);$

$\vdots$

$U := \tau(U, x_j);$

$U := \tau(U, 0);$

$\vdots$

$U := \tau(U, 0);$

$\Big\}$ $(k - j)$-krát

modifikace programu $P_e$

$x_1 := \pi_{k1}(U)$

**end**

# počítání funkcí s omezeným počtem proměnných

modifikace programu $P_e$:

$x_i := x_l + 1$

**begin**
$V := \pi_{kl}(U)$;
$V := V + 1$;
$W := \pi_{k1}(U)$;
$W := \tau(W, \pi_{k2}(U))$;
$W := \tau(W, \pi_{k3}(U))$;
$\vdots$
$W := \tau(W, V)$; // $V$ místo $\pi_{ki}(U)$
$\vdots$
$W := \tau(W, \pi_{kk}(U))$;
$U := W$
**end**

# počítání funkcí s omezeným počtem proměnných

modifikace programu $P_e$:

**while** $x_i \neq x_l$ **do** $\delta$

**begin**
  $V := \pi_{ki}(U)$;
  $W := \pi_{kl}(U)$;
  **while** $V \neq W$ **do begin**
    modifikovaná verze $\delta$;
    $V := \pi_{ki}(U)$;
    $W := \pi_{kl}(U)$
  **end**
**end**

- program $P_{short(e)}$ počítá tutéž $j$-arní funkci jako $P_e$
- funkce *short* spočítá z indexu programu $P_e$ index programu $P_{short(e)}$, tj. provede zde uvedenou konstrukci

■

# překlad programu na čtveřice

další úskalí implementace univerzální funkce $\Phi$

- není snadno patrné, v jakém pořadí instrukce vykonávat
  (Pokračovat další instrukcí nebo se vrátit k testu?
  Ke kterému testu?)
- program proto převedeme na čtveřice
  (návěští, instrukce, návěští pro false, návěští pro true)

**begin**
  **while** $x_1 \neq x_2$ **do**
    **while** $x_1 \neq x_4$ **do**
      $x_1 := x_1 + 1;$
  $x_2 := x_4$
**end**

$\longrightarrow$

$(1,\ x_1 \neq x_2,\ 4,\ 2)$
$(2,\ x_1 \neq x_4,\ 1,\ 3)$
$(3,\ x_1 := x_1 + 1,\ 2,\ 2)$
$(4,\ x_2 := x_4,\ 5,\ 5)$

- konečné posloupnosti čtveřic lze kódovat přirozenými čísly

# funkce pro práci se čveřicemi

**Lemma 5.19**

*Existuje totálně vyčíslitelná funkce* $quad : \mathbb{N} \mapsto \mathbb{N}$ *taková, že* $quad(e)$ *je kód posloupnosti čtveřic reprezentující program* $P_e$.

Nechť *LIST* je kód posloupnosti čtveřic. Důkaz věty o numeraci používá i tyto totálně vyčíslitelné funkce:

$max(LIST)$ vrácí nejvyšší návěští v *LIST* (tj. konec programu)

$fetch(LIST, LBL)$ najde v *LIST* čtveřici s návěštím *LBL* a vrátí kód odpovídající instrukce

$next(LIST, LBL, BLN)$ najde v *LIST* čtveřici s návěštím *LBL* a vrátí návěští pro false pokud $BLN = 0$ nebo návěští pro true pokud $BLN = 1$

# důkaz věty o numeraci

**Důkaz:** program počítající univerzální funkci $\Phi : \mathbb{N}^{j+1} \to \mathbb{N}$:

**begin**
  $E := short(x_1)$;  $x_1 := x_2$;  $x_2 := x_3$;  ... $x_j := x_{j+1}$;  $x_{j+1} := 0$;
  $LIST := quad(E)$;
  $LBL := 1$;
  $MAX := max(LIST)$;
  **while** $LBL \neq MAX$ **do begin**
    $INST := fetch(LIST, LBL)$;
    $BLN := 0$;
    **if** $INST = code(x_1 := 0)$ **then** $x_1 := 0$;

      .
      .
      .
    **if** $INST = code(x_k := 0)$ **then** $x_k := 0$;
    **if** $INST = code(x_1 := x_1 + 1)$ **then** $x_1 := x_1 + 1$;
    **if** $INST = code(x_1 := x_1 - 1)$ **then** $x_1 := x_1 - 1$;
    **if** $INST = code(x_1 := x_2 + 1)$ **then** $x_1 := x_2 + 1$;
    **if** $INST = code(x_1 := x_2 - 1)$ **then** $x_1 := x_2 - 1$;

      .
      .
      .
    **if** $INST = code(x_k := x_k + 1)$ **then** $x_k := x_k + 1$;
    **if** $INST = code(x_k := x_k - 1)$ **then** $x_k := x_k - 1$;
    **if** $INST = code(x_1 \neq x_2) \wedge x_1 \neq x_2$ **then** $BLN := 1$;
    **if** $INST = code(x_1 \neq x_3) \wedge x_1 \neq x_3$ **then** $BLN := 1$;

      .
      .
      .
    **if** $INST = code(x_{k-1} \neq x_k) \wedge x_{k-1} \neq x_k$ **then** $BLN := 1$;
    $LBL := next(LIST, LBL, BLN)$;
  **end**
**end**