IB107 Vyčíslitelnost a složitost

úvod, while-programy, makropříkazy, Churchova teze

Jan Strejček

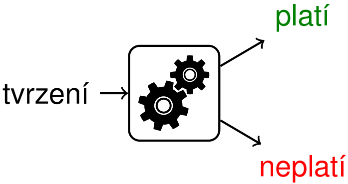
Fakulta informatiky
Masarykova univerzita

trocha historie

Hilbertův program

- formulovat konečnou množinu axiomů tak, aby z nich pomocí odvozovacích pravidel bylo možné získat právě veškerá pravdivá matematická tvrzení

David Hilbert



Kurt Gödel

první Gödelova věta o neúplnosti (1931)

- taková množina axiomů neexistuje ani pro tvrzení o aritmetice přirozených čísel se sčítáním a násobením

základní otázky

vyčíslitelnost

- Které problémy jsou algoritmicky řešitelné a které ne?
- Co to vlastně je algoritmus?
- Jaké jsou důvody neexistence algoritmů pro řešení některých problémů?

složitost

- Jsou všechny algoritmicky řešitelné problémy stejně těžké?
- Jaká je složitost algoritmu?
- Co je to složitost problému?

algoritmus

požadavky

- konečný zápis
- vykonává se mechanicky
- jednotlivé kroky algoritmu se vykonávají diskrétně

formalismy pro popis algoritmů

- Turingův stroj
- λ-kalkul
- Postovy systémy
- C++, python, ...
- while-programy
- ...

notace

- $\mathbb{N} = \{0, 1, 2, \ldots\}$
- $\mathbb{R}_0^+ =$ nezáporná reálná čísla

- částečné funkce (neboli zobrazení) $f : X \to Y$
- definiční obor $dom(f)$
- obor hodnot $range(f)$
- nedefinováno $\perp$
- totální funkce $f : X \mapsto Y$

- obraz množiny $A \subseteq \mathbb{N}^k$ při zobrazení $f : \mathbb{N}^k \to \mathbb{N}$

$$f(A) = \{f(a) \mid a \in dom(f) \cap A\}$$

- vzor množiny $B \subseteq \mathbb{N}$ při zobrazení $f : \mathbb{N}^k \to \mathbb{N}$

$$f^{-1}(B) = \{a \mid a \in dom(f) \wedge f(a) \in B\}$$

syntaxe while-programů

$$
\begin{array}{lcl}
\langle \mathit{var} \rangle & \in & \mathit{Var} = \{x, y, z, \ldots, x_1, x_2, x_3, \ldots\} \\
\langle \mathit{asgt} \rangle & \rightarrow & \langle \mathit{var} \rangle := 0 \mid \langle \mathit{var} \rangle := \langle \mathit{var} \rangle + 1 \mid \langle \mathit{var} \rangle := \langle \mathit{var} \rangle - 1 \\
\langle \mathit{stm} \rangle & \rightarrow & \langle \mathit{asgt} \rangle \mid \underline{\textbf{while}}\ \langle \mathit{var} \rangle \neq \langle \mathit{var} \rangle\ \underline{\textbf{do}}\ \langle \mathit{stm} \rangle \mid \langle \mathit{prg} \rangle \\
\langle \mathit{seq} \rangle & \rightarrow & \langle \mathit{stm} \rangle \mid \langle \mathit{seq} \rangle;\ \langle \mathit{stm} \rangle \\
\langle \mathit{prg} \rangle & \rightarrow & \underline{\textbf{begin}}\ \ \underline{\textbf{end}} \mid \underline{\textbf{begin}}\ \langle \mathit{seq} \rangle\ \underline{\textbf{end}}
\end{array}
$$

$\mathbb{P}$ je množina všech programů.

sémantika while-programů

intuitivně

- proměnné nabývají hodnot z $\mathbb{N}$
- během běhu programu se hodnoty proměnných mění
- výsledkem odečítání do záporu je 0
- zajímá nás, jak program změní počáteční hodnoty na hodnoty po doběhnutí programu (pokud program skončí)

formálně

- stav $\sigma : \mathit{Var} \mapsto \mathbb{N}$
- *Env* je množina všech stavů
- sémantika programu P je dána částečnou funkcí $[\![P]\!] : \mathit{Env} \to \mathit{Env}$

sémantika while-programů

- modifikace jedné proměnné stavu

$$\sigma[x \leftarrow a](y) = \begin{cases} \sigma(y) & \text{pokud } y \neq x \\ a & \text{pokud } y = x \end{cases}$$

- n-násobná kompozice funkce f

$$\begin{aligned} f^0(x) &\stackrel{\text{df}}{=} x \\ f^{n+1}(x) &\stackrel{\text{df}}{=} f(f^n(x)) \end{aligned}$$

- odečítání na $\mathbb{N}$

$$x \ominus y = \begin{cases} x - y & \text{pokud } x \geq y \\ 0 & \text{jinak} \end{cases}$$

sémantika while-programů

$$\begin{array}{rcl}
[\![x := 0]\!](\sigma) & \stackrel{\text{df}}{=} & \sigma[x \leftarrow 0] \\
[\![x := y + 1]\!](\sigma) & \stackrel{\text{df}}{=} & \sigma[x \leftarrow \sigma(y) + 1] \\
[\![x := y - 1]\!](\sigma) & \stackrel{\text{df}}{=} & \sigma[x \leftarrow \sigma(y) \ominus 1] \\
[\![\underline{\textbf{while}}\ x \neq y\ \underline{\textbf{do}}\ \delta]\!](\sigma) & \stackrel{\text{df}}{=} & \begin{cases} [\![\delta]\!]^n(\sigma) & \text{kde } n \text{ je nejmenší číslo takové, že } [\![\delta]\!]^n(\sigma) \text{ je definováno a } [\![\delta]\!]^n(\sigma)(x) = [\![\delta]\!]^n(\sigma)(y) \\ \perp & \text{pokud takové } n \text{ neexistuje} \end{cases} \\
[\![\delta_1;\ \delta_2]\!](\sigma) & \stackrel{\text{df}}{=} & [\![\delta_2]\!]([\![\delta_1]\!](\sigma)) \\
[\![\underline{\textbf{begin}}\ \ \underline{\textbf{end}}]\!](\sigma) & \stackrel{\text{df}}{=} & \sigma \\
[\![\underline{\textbf{begin}}\ \delta\ \underline{\textbf{end}}]\!](\sigma) & \stackrel{\text{df}}{=} & [\![\delta]\!](\sigma) \\
\\
\text{přičemž } [\![\delta]\!](\perp) & = & \perp
\end{array}$$

sémantická funkce programu

Definice 4.1 (sémantická funkce programu)

Nechť P je while-program a $j \geq 0$. Sémantická funkce (arity j) programu P je funkce $\varphi_P^{(j)} : \mathbb{N}^j \to \mathbb{N}$ definovaná jako

$$\varphi_P^{(j)}(a_1, \ldots, a_j) = \begin{cases} [\![P]\!](\sigma)(x_1) & \textit{je-li } [\![P]\!](\sigma) \textit{ definováno} \\ \perp & \textit{jinak} \end{cases}$$

kde σ je počáteční stav definovaný takto:

$$\sigma(x) = \begin{cases} a_i & \textit{pokud } x = x_i \textit{ pro } 1 \leq i \leq j \\ 0 & \textit{jinak} \end{cases}$$

Horní index $^{(j)}$ vynecháváme, je-li zřejmý z kontextu.

příklad

begin
  while $x_2 \neq x_3$ **do** **begin**
    $x_1 := x_1 + 1;$
    $x_2 := x_2 - 1$
  end
end

begin
  begin
  end
end

vyčíslitelné funkce

Definice 4.2 (vyčíslitelná funkce)

Funkce $\Psi : \mathbb{N}^j \to \mathbb{N}$ *je (efektivně) vyčíslitelná, právě když existuje while-program P takový, že*

$$\Psi = \varphi_P^{(j)}.$$

Je-li Ψ *totální a vyčíslitelná, pak se nazývá totálně vyčíslitelná.*

$\mathcal{P}^{(j)}$ je množina všech j-árních vyčíslitelných funkcí,
množinu unárních vyčíslitelných funkcí značíme také $\mathcal{P}$.

VF = vyčíslitelná funkce
TVF = totálně vyčíslitelná funkce

makropříkazy - rozšíření syntaxe while-programů

$z := x$ **begin** $z := x + 1;\ z := z - 1$ **end**

$z := z + x$

```
begin
   u := 0;
   while u ≠ x do begin
            z := z + 1;
            u := u + 1
   end
end
```

$z := x + y$ **begin** $z := x;\ z := z + y$ **end**

makropříkazy - rozšíření syntaxe while-programů

Domluva

- Makropříkazy nemění hodnoty vstupních proměnných (s výjimkou situací, kdy je změna explicitně požadována).
- Implementace makropříkazů používá jiné proměnné než program obsahující makropříkaz.

$z := n$ pro libovolné $n \in \mathbb{N}$

$z := x - y$

$z := x * y$

$z := x \text{ div } y$

$z := x \text{ mod } y$

$z := x^y$

$\vdots$

makropříkazy pro testy

Nechť α, β jsou proměnné nebo čísla.

1. Každý výraz tvaru $\alpha < \beta$, $\alpha \neq \beta$, $\alpha = \beta$ je test.
2. Jsou-li T_1 a T_2 testy, pak i $T_1 \wedge T_2$, $T_1 \vee T_2$ a $\neg T_1$ jsou testy.

Tvrzení 4.4

Pro libovolný test T a libovolné δ je **while** T **do** δ *makropříkaz jazyka while-programů.*

Důkaz:

- čísla v testu lze nahradit proměnnými se stejnou hodnotou
- dále předpokládáme, že test T neobsahuje čísla
- pro test T vytvoříme aritmetický výraz E_T, který má hodnotu 1, právě když T je pravdivé (a jinak má hodnotu 0)

makropříkazy pro testy

T	E_T
$x < y$	$(y - x) - (y - (x + 1))$
$x \neq y$	
$x = y$	
$T_1 \wedge T_2$	$(E_{T_1} + E_{T_2}) - 1$
$T_1 \vee T_2$	$(E_{T_1} + E_{T_2}) - ((E_{T_1} + E_{T_2}) - 1)$
$\neg T_1$	$1 - E_{T_1}$

while T **do** δ

begin
  $u := E_T$;
  $v := 0$;
  while $u \neq v$ **do** **begin** δ; $u := E_T$ **end**
end

další makropříkazy

Tvrzení 4.5

Pro libovolný test T a libovolné $\delta, \delta_1, \delta_2$ jsou příkazy

- $\underline{\textbf{if}}\ T\ \underline{\textbf{then}}\ \delta$
- $\underline{\textbf{if}}\ T\ \underline{\textbf{then}}\ \delta_1\ \underline{\textbf{else}}\ \delta_2$
- $\underline{\textbf{repeat}}\ \delta\ \underline{\textbf{until}}\ T$

makropříkazy jazyka while-programů.

Důkaz:

$\underline{\textbf{if}}\ T\ \underline{\textbf{then}}\ \delta$

$\underline{\textbf{begin}}$
$\quad v := 0;$
$\quad \underline{\textbf{while}}\ T \wedge (v = 0)\ \underline{\textbf{do}}\ \underline{\textbf{begin}}\ \delta;\ v := 1\ \underline{\textbf{end}}$
$\underline{\textbf{end}}$

$\blacksquare$

Churchova teze

- Vyčíslitelné funkce jsou jistě algoritmické.
- Existuje nějaký chytřejší formalismus pro popis algoritmů, který by uměl popsat algoritmy pro více funkcí?
- Odpovídá pojem vyčíslitelné funkce našemu očekávání?

Churchova teze

Každá částečná funkce nad přirozenými čísly je intuitivně vyčíslitelná (v jakémkoliv akceptovatelném smyslu) tehdy a jen tehdy, když je vyčíslitelná while-programem.

- nelze dokázat, lze jen vyvrátit
- “důkazy Churchovou tezí”