Program, algoritmus, funkce

Program je formální popis chování nějakého systému (aplikační program, reaktivní systém, operační systém. . .). Skládá se z popisu algoritmů.

Algoritmus lze chápat jako parciální funkci z množiny všech možných vstupů do množiny všech možných výstupů.

$$\text{funkce} : \text{Vstupy} \dashrightarrow \text{Výstupy}$$

Pozor, není to definice: sice každému algoritmu odpovídá parciální funkce ze vstupů do výstupů, ale naopak ne každou takovou funkci lze vyjádřit algoritmem.

Algoritmus je parciální funkce z množiny vstupů do množiny výstupů, k níž existuje konečný popis v určitém formalismu.

Tímto formalismem je typicky programovací jazyk, ale existují i další matematické formalismy: Turingův stroj, RAM, lambda kalkul, kombinátorový kalkul, vyčíslitelné funkce, Markovův algoritmus . . .

Slovo *algoritmus* je odvozeno ze jména starověkého perského matematika a astronoma al-Chwarezmího (Abū ‘Abd Allāh Muḥammad ibn Mūsā al-Khwārizmī).

Funkce z množiny vstupů do množiny výstupů, které lze vyjádřit algoritmem, se nazývají *rekursivně spočetné*.

Reaktivní systém je program, jehož provádění normálně nemusí končit (např. operační systémy apod.), v obecnějším smyslu jsou to interaktivní programy.

Program, který je implementovaným algoritmem, při výpočtu přečte vstup, zpracuje ho (vstupní data transformuje na výstupní), vypíše výstup a skončí.

Většina současných programů je interaktivních, ale součástí všech jsou algoritmy.

Korektnost algoritmu

In	množina vstupních dat
Out	množina výstupních dat
$A : \mathit{In} \dashrightarrow \mathit{Out}$	algoritmus (na vstupu z *In* vypisující výstup z *Out*)
$\varphi : \mathit{In} \rightarrow \mathit{Bool}$	vstupní podmínka
$\psi : \mathit{In} \times \mathit{Out} \rightarrow \mathit{Bool}$	výstupní podmínka

Vstupní podmínka φ určuje, zda daný vstupní údaj je pro algoritmus *přípustný*, vymezuje tedy jistou podmnožinu množiny vstupních dat *In*.

Výstupní podmínka ψ říká, zda daný výstupní údaj je či není výsledkem výpočtu algoritmu na daném vstupním údaji.

Vstupní podmínka je unární predikát na množině vstupních dat, výstupní podmínka je binární predikát na množinách vstupních a výstupních dat.
Výstupní podmínka ψ říká, zda daný výstupní údaj je či není (pro nedeterministické algoritmy zda může či nemůže být) výsledkem výpočtu algoritmu na daném vstupním údaji.

Příklad: Algoritmus A přečte tři čísla a vypíše je v neklesajícím pořadí; funguje korektně pro „malá" přirozená čísla.

In $=$ *Out* $= \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$,

$\varphi_A(x, y, z) \equiv 0 \leq x < 2^{31} \wedge 0 \leq y < 2^{31} \wedge 0 \leq z < 2^{31}$

$\psi_A(x, y, z, u, v, w) \equiv (u, v, w) \in \mathit{Permut}(x, y, z) \wedge u \leq v \leq w$

Poznámka: Připustíme i algoritmy, které nenačítají žádný vstup. Pro ně položíme *In* $= \{\emptyset\}$.

Def: Řekneme, že A je *konvergentní* vzhledem k φ, když množina $\{x \in \mathit{In} \mid \varphi(x)\}$ je podmnožinou definičního oboru funkce A, tj. když pro každou vstupní hodnotu x, pro niž platí $\varphi(x)$, je výsledek výpočtu podle algoritmu A definován (výpočet se zastaví).

Def: Řekneme, že A je *parciálně korektní* vzhledem k φ a ψ, když pro každou vstupní hodnotu x z definičního oboru funkce A splňující vstupní podmínku φ (tj. pro každé x, pro něž je $A(x)$ definováno a $\varphi(x)$) platí $\psi(x, A(x))$.

Def: Řekneme, že A je *totálně korektní* vzhledem k φ a ψ, když je konvergentní vzhledem k φ a parciálně korektní vzhledem k φ a ψ.

Důsledek: Algoritmus A je totálně korektní vzhledem k φ a ψ, právě když pro každou vstupní hodnotu x splňující vstupní podmínku výpočet podle algorimu A skončí a jeho výsledek splňuje i výstupní podmínku.

Vstupní a výstupní podmínce se také říká *specifikace algoritmu*. O totálně korektním algoritmu pak říkáme, že *vyhovuje specifikaci*.

Výpočetní paradigmata

Paradigma (vzor, přístup, pojetí) udává způsob, jakým je algoritmus vyjádřen.

Výpočetní paradigmata:

– imperativní

– funkcionální

– logické

– procedurální

– objektové

– sekvenční

– paralelní

⋮

Imperativní paradigma

Neformální definice:

Výpočet je posloupností tzv. *výpočetních kroků*. Výpočetní krok je realizací instrukce (jednoduchého příkazu) programu. Přenos informace mezi jednotlivými kroky výpočtu zprostředkovává *stav*.

Jinak řečeno, příkazy programu jsou *stavové transformátory* — (parciální) funkce z množiny stavů do množiny stavů.

Pro formální definici je nutno zavést abstraktní počítač. V sekvenčním imperativním paradigmatu jím nejčastěji bývá Turingův stroj nebo RAM (Random-Access Machine).

Může jím být také „vyšší“ imperativní jazyk, který kromě elementárních stavových transformátorů (typicky tzv. *přiřazovacího příkazu*) obsahuje složené stavové transformátory realizující sekvenci (begin-end), binární větvení výpočtu (if-then-else), násobné větvení (case), opakování (while-do, for-do), apod.

Výpočetním krokem je pak přechod mezi dvěma konfiguracemi Turingova stroje, resp. provedení jedné instrukce RAMu, resp. provedení jednoho *elementárního* příkazu.

Abstraktní počítače pro imperativní programy

- Turingův stroj
- RAM
- (abstraktní) programovací jazyk

  ⋮

Turingův stroj

- Konečná neprázdná množina Σ symbolů — tzv. *abeceda*
- Nekonečná *páska* — spočetná posloupnost políček. Každé políčko pásky obsahuje jeden symbol abecedy, přičemž skoro všechna políčka obsahují zvláštní symbol $\sqcup$.
- *Čtecí hlava*, jejíž pozice na pásce je dána přirozeným číslem.
- Konečná množina S vnitřních stavů s vyznačeným *počátečním stavem* σ_0 a *koncovými stavy* $F \subseteq S$.
- *Přechodová funkce* $\delta : S \times \Sigma \to S \times \Sigma \times \{\mathtt{L}, \mathtt{R}\}$

RAM

- Spočetná posloupnost $\mathcal{M}$ registrů (paměťových míst) — tzv. *paměť*. Každý registr obsahuje celé číslo, vždy však skoro všechny obsahují nulu.
- *Stav* $\sigma \in S = \mathbb{N} \times \mathbb{Z}^*$
- Konečná množina instrukcí $\mathcal{I}$. Každá instrukce $\iota \in \mathcal{I}$ má danou sémantiku — funkci $\iota : S \dashrightarrow S$.
- Konečná posloupnost $P \in \mathcal{I}^*$ instrukcí, tzv. *program*

Příklad: Výpočet faktoriálu (funkcionální verze – Pascal)

Faktoriál nezáporného celého čísla n je číslo $n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-1) \cdot n$.
Speciálně $0! = 1$ (součin nulového počtu činitelů).

```
function fact (n:Integer): Integer;
begin
 if n==0 then fact:= 1             {return 1}
         else fact:= n*fact(n-1) {return n*fact(n-1)}
end
```

Věta: Funkce `fact` konverguje vzhledem ke vstupní podmínce $\varphi(n) \equiv n \in \mathbb{N}$.

Věta: Funkce `fact` je parciálně korektní vzhledem k φ a k výstupní podmínce $\psi(n, k) \equiv k = n!$.

Důsledek: `fact` je totálně korektní vzhledem k φ, ψ.

Věta: Funkce `fact` konverguje vzhledem ke vstupní podmínce $\varphi(n) \equiv n \in \mathbb{N}$.

Důkaz indukcí podle n. Pro $n = 0$ se výpočet zastaví; zastaví-li se výpočet pro n, zastaví se i pro $n + 1$.

Věta: Funkce `fact` je parciálně korektní vzhledem k φ a k výstupní podmínce $\psi(n, k) \equiv k = n!$.

Důkaz opět indukcí podle n. $\texttt{fact}(0) = 1 = 0!$.
Nechť $n > 0$ a tvrzení platí pro $n - 1$. Pak
$\texttt{fact}(n) = n \cdot \texttt{fact}(n - 1) = n \cdot (n - 1)! = n!$, tedy tvrzení platí i pro n.

Výpočet faktoriálu (imperativní verze – Pascal)

```
function factorial (n:Integer): Integer;
var i, k: Integer;
begin
  i := 0;
  k := 1;
  while  i < n  do
   begin i := i+1;
         k := k*i
   end;
  factorial:= k    {return k}
end
```

Věta: Funkce konverguje vzhledem ke vstupní podmínce $\varphi(n) \equiv n \in \mathbb{N}$.

Funkce je parciálně korektní vzhledem k φ a k výstupní podmínce $\psi(n,k) \equiv k = n!$.

Věta: Funkce `factorial` konverguje vzhledem ke vstupní podmínce $\varphi(n) \equiv n \in \mathbb{N}$.

Důkaz spočívá v nalezení číselné hodnoty (závislé na obsahu programových proměnných), která v žádném okamžiku výpočtu nemůže být záporná a přitom při každém průchodu tělem cyklu klesne nejméně o jedničku. V tomto příkladě je touto hodnotou číslo $n - i$ a důkaz nezápornosti i klesání je snadný.

Věta: Funkce `factorial` je parciálně korektní vzhledem k φ a k výstupní podmínce $\psi(n,k) \equiv k = n!$.

Důkaz spočívá v nalezení mezilehlých podmínek, zejména invariantu cyklu while. Invariant cyklu v místě testování podmínky je $k = i! \ \wedge \ 0 \leq i \leq n$.

Invariant cyklu

je mezilehlá podmínka, která je splněna v daném bodě výpočtu podle algoritmu Invariant cykluv každém průchodu cyklem.

V našem příkladě bude invariantem cyklu while podmínka $k = i! \wedge 0 \leq i \leq n$ a bude se vztahovat k místu testování podmínky cyklu. Důkaz parciální korektnosti funkce `factorial` se rozloží do kroků:

1. Platí-li na začátku $\varphi(n)$, pak po provedení úvodních přiřazení bude splněn invariant cyklu.

2. Je-li v nějakém okamžiku splněn invariant cyklu a provede se tělo cyklu, pak po jeho provedení bude opět splněn invariant.

3. Je-li splněn invariant cyklu a není splněna podmínka cyklu, pak je splněna mezilehlá podmínka za cyklem, tedy $\psi(n, k)$.

Pro místa v algoritmu ležící mezi příkazy stanovíme tzv. *mezilehlé podmínky*. Podmínka na začátku bude $\varphi(n)$, podmínka na konci bude $\psi(n, k)$. Z ostatních bodů v algoritmu pro důkaz parciální korektnosti obvykle stačí vybrat jen význačná místa, v nichž se stav výpočtu významně mění. Takovými místy jsou body uvnitř cyklů, například na jejich začátku. Mezilehlá podmínka pro místo uvnitř cyklu se nazývá *invariant cyklu*.

Příklad: Algoritmus umocňování čísla na přirozený exponent (funkcionální verze – Pascal)

```
function pow (z:Real; n:Integer): Real;
{ Předpokládá se z>0, n≥0 }
begin
  if n==0 then pow := 1
          else pow := z * pow(z,n-1)
end
```

Věta: Funkce pow konverguje vzhledem ke vstupní podmínce $\varphi(z,n) \equiv z \in \mathbb{R}, n \in \mathbb{N}, z > 0$.

Důkaz: Klesající hodnotou je zde exponent.

Věta: Funkce pow je parciálně korektní vzhledem ke vstupní podmínce φ a k výstupní podmínce $\psi(z,n,r) \;\equiv\; r = z^n$.

Důkaz věty: Indukcí podle exponentu.

Příklad: Algoritmus umocňování čísla půlením exponentu (imperativní verze – Pascal)

```
function power (z:Real; n:Integer): Real;
{ Předpokládá se z>0, n≥0 }
var y,r:Real; k:Integer;
begin
  k := n;
  y := z;
  r := 1;
  while k > 0 do
    begin
      if odd(k) then r := r * y;
      k := k div 2;
      y := sqr(y)
    end;
  power := r
end
```

Věta: Funkce `power` konverguje vzhledem ke vstupní podmínce $\varphi(z,n) \equiv z \in \mathbb{R}, n \in \mathbb{N}, z > 0$.

Důkaz: Klesající hodnotou je zde exponent. Klesání je tentokrát rychlejší než v předcházejícím příkladě, ale pro $\mathtt{k} > 0$ je vždy ostré.

Věta: Funkce `power` je parciálně korektní vzhledem ke vstupní podmínce φ a k výstupní podmínce $\psi(z,n,r) \equiv r = z^n$.

Invariant: $\mathtt{r} \cdot \mathtt{y}^{\mathtt{k}} = z^n \ \wedge \ \mathtt{k} \geq 0$

Věta: Funkce `power` konverguje vzhledem ke vstupní podmínce $\varphi(z,n) \equiv z \in \mathbb{R}, n \in \mathbb{N}, z > 0$.

Důkaz: Klesající hodnotou je zde exponent. Klesání je tentokrát rychlejší než v předcházejícím příkladě, ale pro $\mathtt{k} > 0$ je vždy ostré.

Věta: Funkce `power` je parciálně korektní vzhledem ke vstupní podmínce φ a k výstupní podmínce $\psi(z,n,r) \equiv r = z^n$.

Důkaz: Invariant cyklu while, který platí vždy v okamžiku testování podmínky cyklu ($\mathtt{k} > 0$), je

$$\mathtt{r} \cdot \mathtt{y}^{\mathtt{k}} = z^n \;\wedge\; \mathtt{k} \geq 0$$

Po poslední iteraci platí tento invariant v konjunkci spolu s negací podmínky cyklu ($\neg(\mathtt{k} > 0)$). Z této konjunkce vyplývá $\mathtt{r} = z^n$, přičemž $\mathtt{r}$ je výsledná hodnota funkce `power`. Platí tedy i výstupní podmínka $\psi(z, n, \mathtt{power(}z\mathtt{,}n\mathtt{)})$.

Příklad: Algoritmus řazení vkládáním (imperativní verze – Pascal)

```
const MAX = 999;
type Elem = Integer;
     Posl = array [1..MAX] of Elem;
procedure iInsSort (n:Integer; var A:Posl);
  var i, j : Integer;  x : Elem;
  begin
      for i := 2 to n do
      begin
          x := A[i];  j := i-1;
          while (j>0) && (A[j]>x) do
          begin
              A[j+1] := A[j];
              j := j-1
          end;
          A[j+1] := x
       end
   end
```

Věta: Algoritmus `iInsSort` je totálně korektní vzhledem k podmínkám

$\varphi(n, A) \equiv$ posloupnost A je neprázdná a $n \geq 1$ je její délka

$\psi(n, A, A') \equiv$ posloupnost A' je permutací posloupnosti A a je neklesající

Věta: Nechť $\mathbb{Z}^*$ označuje množinu všech konečných celočíselných posloupností, *In* $= \mathbb{N} \times \mathbb{Z}^*$,*Out* $= \mathbb{Z}^*$. Nechť $\varphi(n, A) \equiv n \geq 1 \wedge |A| = n$, $\psi(n, A, A') \equiv\ A' \in$ *Permut*$(A) \wedge \forall i, j, 1 \leq i < j \leq n. A'_i \leq A'_j$, Pak `iInsSort` je totálně korektní vzhledem k podmínkám φ, ψ.

Důkaz rozložíme na důkaz konvergence a parciální korektnosti.

Lemma: Algoritmus `iInsSort` je konvergentní vzhledem k φ.

Důkaz: Vnější cyklus for vždy skončí, protože je vždy předem dán pevný počet jeho opakování ($n - 1$). Vnitřní cyklus while se provádí jen když $\mathtt{j} > 0$. Ale při každém průchodu tělem tohoto cyklu hodnota proměnné `j` klesne. To znamená, že cyklus while se provede konečněkrát (nejvýše $(\mathtt{i} - 1)$-krát).

Lemma: Algoritmus `iInsSort` je parciálně korektní vzhledem k podmínkám φ, ψ.

Důkaz: Vstupní posloupnost označme $(a_1, \ldots, a_n)$, obsah pole `A` (v daném stavu výpočtu) označme $(A_1, \ldots, A_n)$.

Invariantem vnějšího cyklu for (splněným vždy na začátku tohoto cyklu) je podmínka $A_1 \leq A_2 \leq \cdots \leq A_{i-1} \ \wedge \ (A_1, \ldots, A_{i-1}) \in \textit{Permut}(a_1, \ldots, a_{i-1}) \ \wedge \ A_i = a_i, \ldots, A_n = a_n$, takže po posledním průchodu cyklem for platí $A_1 \leq A_2 \leq \ldots \leq A_n$.

Invariantem vnitřního cyklu while (opět splněným vždy na začátku cyklu) je podmínka $A_1 \leq \cdots \leq A_{j-1} \leq A_j = A_{j+1} \leq A_{j+2} \leq \cdots \leq A_i \ \wedge \ (A_1, \ldots, A_j, \ a_i, \ A_{j+2}, \ldots, A_i) \in \textit{Permut}(a_1, \ldots, a_i)$

(Důkaz, že jde skutečně o invarianty, je snadný.)

Funkcionální paradigma

Neformální definice:

Výpočet je posloupností tzv. *redukčních kroků*, z nichž každý je elementární úpravou programu (výrazu). Výsledkem výpočtu je hodnota, kterou už nelze dále zjednodušit.

Délka výpočtu je pak délkou této posloupnosti, tj. počet redukčních kroků vedoucích k výsledku.

Pro formální definici je nutno zavést formální kalkul. Ve funkcionálním paradigmatu jím nejčastěji bývá jazyk vycházející z lambda kalkulu — *funkcionální jazyk*.

Příklad: Funkce `maxim` najde největší číslo z konečného neprázdného seznamu čísel

```
maxim [x]       =  x
maxim (x:y:s)   =  if x>y then maxim (x:s)
                          else maxim (y:s)
```

```
 maxim [4,3,5,1]
⇝ if 4>3  then maxim [4,5,1] else maxim [3,5,1]
⇝ if True then maxim [4,5,1] else maxim [3,5,1]

⇝ maxim [4,5,1]
⇝ if 4>5   then maxim [4,1] else maxim [5,1]
⇝ if False then maxim [4,1] else maxim [5,1]

⇝ maxim [5,1]
⇝ if 5>1  then maxim [5] else maxim [1]
⇝ if True then maxim [5] else maxim [1]

⇝ maxim [5]

⇝ 5
```

Příklad: Algoritmus řazení vkládáním (funkcionální verze – Haskell)

```
fInsSort []    =  []
fInsSort (x:s) =  ins x (fInsSort s)
  where ins x []    = [x]
        ins x (y:t) = if x<=y then x:(y:t)
                              else y:(ins x t)
```

Nechť *In* $=$ *Out* je množina všech seznamů (posloupností) celých čísel,

$\varphi(s) \equiv s$ je konečný,

$\psi(s,t) \equiv$ „t je permutací seznamu s a t je neklesající“.

Věta: AlgoritmusPak `fInsSort` je totálně korektní vzhledem k φ, ψ.

Poznámka: V Haskellu lze definici zapsat i kratčeji:

`fInsSort = foldr ins []` .

Věta: AlgoritmusPak `fInsSort` je totálně korektní vzhledem k φ, ψ.

Důkaz: Nechť s je konečný seznam délky n. Konvergence obou funkcí se ukáže indukcí přes délku seznamu.

Při důkazu parciální korektnosti vyjdeme z vlastnosti funkce `ins`: Je-li seznam u neklesající, pak `ins` y u je neklesající permutace seznamu y`:`u. Zbytek indukcí přes délku seznamu s.