Program, algoritmus, funkce

Program je formální popis chování jakéhokoli 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 parciÆlní funkce z množiny vstupů do množinyvýstupů, k níž existuje konečný popis v určitØm formalismu.

Tímto formalismem je typicky programovací jazyk, ale existují i další matematickØ formalismy: Turingův stroj, RAM, lambda kalkul, kombinÆtorový kalkul, vyčíslitelnØ funkce, Markovův algoritmus . . .

Slovo *algoritmus* je odvozeno ze jmØna starověkØho perskØho matematika a astronoma al-Chwarezmího (Abū ‘Abd Allāh Muḥammad ibn Mūsā al-Khwārizmī).

Funkce z množiny vstupů do množiny výstupů, kterØ lze vyjÆdřit algoritmem, se nazývají *rekursivně spočetnØ*.

Reaktivní systØm je program, jehož provÆdění normÆlně nemusí končit (např. operační systØmy 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 součÆstí všech jsou algoritmy.

Korektnost algoritmu

In	množina vstupních dat
Out	množina výstupních dat
A : *In* $\dashrightarrow$ *Out*	algoritmus (na vstupu z *In* vypisující výstup z *Out*)
φ : *In* $\rightarrow$ *Bool*	vstupní podmínka
ψ : *In* $\times$ *Out* $\rightarrow$ *Bool*	výstupní podmínka

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

Výstupní podmínka ψ říkÆ, zda daný výstupní œdaj jěi není výsledkem výpočtu algoritmu na danØm vstupním œdaji.

Vstupní podmínka je unÆrní predikÆt na množině vstupních dat, výstupní podmínka je binÆrní predikÆt na množinÆch vstupních a výstupních dat.
Výstupní podmínka ψ říkÆ, zda daný výstupní œdaj jëi není (pro nedeterministickØ algoritmy zda může či nemůže být) výsledkem výpočtu algoritmu na danØm vstupním œdaji.

Příklad: Algoritmus A přečte tři čísla a vypíše je v neklesajícím pořadí; funguje korektně pro „malÆ“ přirozenÆ čísla.

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

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

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

PoznÆmka: Připustíme i algoritmy, kterØ nenačítají žÆdný vstup. Pro ně položíme $\mathit{In} = \{\emptyset\}$.

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

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

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

Důsledek: Algoritmus A je totÆlně korektní vzhledem k φ a ψ, prÆvě 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 takØ říkÆ *specifikace algoritmu*. O totÆlně korektním algoritmu pak říkÆme, že *vyhovuje specifikaci*.

Výpočetní paradigmata

Paradigma (vzor, přístup, pojetí) udÆvÆ způsob, jakým je algoritmus vyjÆden.

Výpočetní paradigmata:

– imperativní

– funkcionÆlní

– logickØ

– procedurÆlní

– objektovØ

– sekvenční

– paralelní

⋮

Imperativní paradigma

NeformÆlní definice:

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

Jinak řečeno, příkazy programu jsou *stavovØ transformÆtory* — (parciÆlní) funkce z množiny stavů do množiny stavů.

Pro formÆlní definici je nutno zavØst abstraktní počítač. V sekvenčním imperativním paradigmatu jím nejčastěji bývÆ Turingův stroj nebo RAM (Random-Access Machine).

Může jím být takØ „vyšší“ imperativní jazyk, který kromě elementÆrních stavových transformÆtorů (typicky tzv. *přiřazovacího příkazu*) obsahuje složenØ stavovØ transformÆtory realizující sekvenci (begin-end), binÆrní větvení výpočtu (if-then-else), nÆsobnØ větvení (case), opakovÆní (while-do, for-do), apod.

Výpočetním krokem je pak přechod mezi dvěma konfiguracemi Turingova stroje, resp. provedení jednØ instrukce RAMu, resp. provedení jednoho *elementÆrního* 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

- SpočetnÆ posloupnost $\mathcal{M}$ registrů (paměťových míst) — tzv. *paměť*. Každý registr obsahuje celØčíslo, vždy však skoro všechny obsahují nulu.
- *Stav* $\sigma \in S = \mathbb{N} \times \mathbb{Z}^*$
- KonečnÆ množina instrukcí $\mathcal{I}$. KaždÆ instrukce $\imath \in \mathcal{I}$ mÆ danou sØmantiku — funkci $\iota : S \dashrightarrow S$.
- KonečnÆ posloupnost $P \in \mathcal{I}^*$ instrukcí, tzv. *program*

Příklad: Výpočet faktoriÁlu (funkcionÁlní verze – Pascal)

FaktoriÁl nezÁpornÝho celÉho čísla n je číslo $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$.

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 faktoriÆlu (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 parciÆlně 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 spočívÆ v nalezenčíselnØ hodnoty (zÆvislØ na obsahu programových proměnných), kterÆ v žÆdnØm okamžiku výpočtu nemůže být zÆpornÆ a přitom při každØm průchodu tělem cyklu klesne nejmØně o jedničku. V tomto příkladě je touto hodnotou číslo $n - i$ a důkaz nezÆpornosti i klesÆní je snadný.

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

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

Invariant cyklu

je mezilehlÆ podmínka, kterÆ je splněna v danØm bodě výpočtu podle algoritmu Invariant cykluv každØm průchodu cyklem.

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

1. Platí-li na začÆtku $\varphi(n)$, pak po provedení œvodních přiřazení bude splněn invariant cyklu.
2. Je-li v nějakØm 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 mezilehlÆ podmínka za cyklem, tedy $\psi(n, k)$.

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

Příklad: Algoritmus umocňovÆní čísla na přirozený exponent (funkcionÆlní 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 parciÆlně 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 umocňovÆnčísla 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. KlesÆní je tentokrÆt rychlejší než v předchÆzejícím příkladě, ale pro $\mathtt{k} > 0$ je vždy ostrØ.

Věta: Funkce `power` je parciÆlně 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. KlesÆní je tentokrÆt rychlejší než v předchÆzejícím příkladě, ale pro $\mathtt{k} > 0$ je vždy ostrØ.

Věta: Funkce `power` je parciÆlně 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 testovÆní 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 tØto konjunkce vyplývÆ $\mathtt{r} = z^n$, přičemž `r` je výslednÆ hodnota funkce `power`. Platí tedy i výstupní podmínka $\psi(z, n, \mathtt{power(}z\mathtt{,}n\mathtt{)})$.

Příklad: Algoritmus řazení vklÆdÆním (imperativní verze – Pascal)

```
1   const MAX = 999;
2   type Elem = Integer;
3        Posl = array [1..MAX] of Elem;
4   procedure iInsSort (n:Integer; var A:Posl);
5     var i, j : Integer;  x : Elem;
6     begin
7         for i := 2 to n do
8         begin
9             x := A[i];  j := i-1;
10            while (j>0) && (A[j]>x) do
11            begin
12                A[j+1] := A[j];
13                j := j-1
14            end;
15            A[j+1] := x
16         end
17     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 $j > 0$. Ale při každém průchodu tělem tohoto cyklu hodnota proměnné `j` klesne. To znamená, že cyklus while se provede konečněkrát (nejvýše $(\texttt{i} - 1)$-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 danØm stavu výpœtu) označme $(A_1, \ldots, A_n)$.
Invariantem vnějšího cyklu for (splněným vždy na začÆtku tohoto cyklu) je podmínka
$A_1 \leq A_2 \leq \cdots \leq A_{i-1} \ \wedge \ (A_1, \ldots, A_{i-1}) \in \mathit{Permut}(a_1, \ldots, a_{i-1}) \ \wedge \ A_i = a_i, \ldots, A_n = a_n$, 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 začÆtku cyklu) je podmínka
$A_1 \leq \cdots \leq A_{j-1} \leq A_j = A_{j+1} \leq A_{j+2} \leq \cdots \leq A_i \ \wedge \ (A_1, \ldots, A_j, \ a_i, \ A_{j+2}, \ldots, A_i) \in \mathit{Permut}(a_1, \ldots, a_i)$

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

FunkcionÆlní paradigma

NeformÆlní definice:

Výpočet je posloupností tzv. *redukčních kroků*, z nichž každý je elementÆrní œpravou programu (výrazu). Výsledkem výpočtu je hodnota, kterou už nelze dÆle zjednodušit.

DØlka výpočtu je pak dØlkou tØto posloupnosti, tj. počet redukčních kroků vedoucích k výsledku.

Pro formÆlní definici je nutno zavØst formÆlní kalkul. Ve funkcionÆlním paradigmatu jím nejčastěji bývÆ jazyk vychÆzející z lambda kalkulu – *funkcionÆlní jazyk*.

Příklad: Funkce `maxim` najde největší číslo z konečnØho neprÆzdnØho seznamčísel

```
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 vklÆdÆním (funkcionÆlní 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 totÆlně korektní vzhledem k φ, ψ.

PoznÆmka: V Haskellu lze definici zapsat i kratčeji:

`fInsSort =  foldr ins []`.

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

Důkaz: Nechť s je konečný seznam dØlky n. Konvergence obou funkcí se ukÆže indukcí přes dØlku seznamu.

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