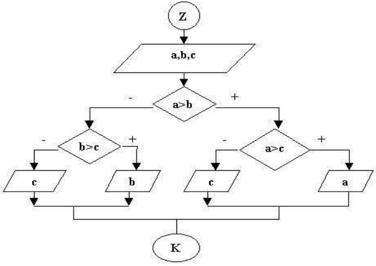
10 Formalizace a důkazy pro algoritmy

Je faktem, že situace, kdy programátorem zapsaný kód ve skutečnosti počítá něco trochu jiného, než si autor představuje, je snad nejčastější programátorskou chybou – o to zákeřnější, že ji žádný „chytrý“ překladač nemůže odhalit.

Proto již na počátku studia informatiky je žádoucí klást důraz na správné chápání zápisu algoritmů i na přesné důkazy jejich vlastností a správnosti.



Stručný přehled lekce

* Jednoduchá formalizace pojmu algoritmus.
* Jak dokazovat vlastnosti a správnost algoritmů.
* Indukce při dokazování algoritmů. Rekurzivní algoritmy.

10.1 Formální popis algoritmu

Před samotným závěrem kurzu si položme otázku, co je vlastně algoritmus?

Poznámka: Za definici algoritmu je obecně přijímána tzv. *Church–Turingova teze* tvrdící, že všechny algoritmy lze „simulovat“ na Turingově stroji. Jedná se sice o přesnou, ale značně nepraktickou definici.

Mimo Turingova stroje existují i jiné *matematické modely výpočtů*, jako třeba stroj RAM, který je abstrakcí skutečného strojového kódu, nebo neprocedurální modely. □

Konvence 10.1. Zjednodušeně zde *algoritmem* rozumíme konečnou posloupnost elementárních výpočetních *kroků*, ve které každý další krok vhodně využívá (neboli závisí na) vstupní údaje či hodnoty vypočtené v předchozích krocích. Tuto závislost přitom pojímáme zcela obecně nejen na operandy, ale i na vykonávané instrukce v krocích.

Pro zápis algoritmu a jeho zpřehlednění a zkrácení využíváme *řídící konstrukce* – podmíněná větvení a cykly. □

Vidíte, jak blízké si jsou konstruktivní matematické důkazy a algoritmy v našem pojetí? Jedná se nakonec o jeden ze záměrů našeho přístupu...

Ukázka algoritmického zápisu

Příklad 10.2. *Zápis algoritmu pro výpočet průměru daného pole* `a[]` *s* `n` *prvky.*

- Inicializujeme `sum` ← `0` ;
- postupně pro `i=0,1,2,...,n-1` provedeme
  - `sum` ← `sum+a[i]` ;
- vypíšeme podíl `(sum/n)` . □

□

Ve „vyšší úrovni“ formálnosti se totéž dá zapsat jako:

Algoritmus 10.3. **Průměr** z daného pole `a[]` s `n` prvky.

```
input pole a[] délky n;
sum ← 0;
foreach  i ← 0,1,2,...,n-1  do
    sum ← sum+a[i];
done
res ← sum/n;
output res;
```

Symbolický zápis algoritmů

Značení. Pro potřeby symbolického formálního zápisu algoritmů v předmětu FI: IB000 si zavedeme následující pravidla:

- *Proměnné* nebudeme deklarovat ani typovat, pole odlišíme závorkami `p[]`.
- *Přiřazení* hodnoty zapisujeme `a` $\leftarrow$ `b`, případně `a := b`, ale nikdy ne `a=b`.
- Jako elem. operace je možné použít jakékoliv *aritmetické výrazy* v běžném matematickém zápise. Rozsahem a přesností čísel se zde nezabýváme. □
- Podmíněné *větvení* uvedeme klíčovými slovy `if ... then ... else ... fi`, kde `else` větev lze vynechat (a někdy, na jednom řádku, i `fi`).
- Pevný *cyklus* uvedeme klíčovými slovy `foreach ... do ... done`, kde část za `foreach` musí obsahovat předem danou množinu hodnot pro přiřazování do řídící proměnné.
- *Podmíněný cyklus* uvedeme klíčovými slovy `while ... do ... done`. Zde se může za `while` vyskytovat jakákoliv logická podmínka. □
- V zápise používáme jasné odsazování (zleva) podle úrovně zanoření řídících struktur (což jsou `if`, `foreach`, `while`).
- Pokud je to dostatečně jasné, elementární operace nebo podmínky můžeme i ve formálním zápise popsat běžným jazykem.

Co počítá následující algoritmus?

Příklad 10.4. *Je dán následující symbolicky zapsaný algoritmus. Co je jeho výstupem v závislosti na vstupech a, b?*

Algoritmus 10.5.

```
input a, b;
res ← 7;
foreach  i ← 1,2,...,b-1,b  do
    res  ←  res+a+2·b+8;
done
output res;
```

□

Vypočítáme hodnoty výsledku res v počátečních iteracích cyklu:

$$\begin{aligned} b = 0&: \quad res = 7, \\ b = 1&: \quad res = 7 + a + 2b + 8, \\ b = 2&: \quad res = 7 + (a + 2b + 8) + (a + 2b + 8), \dots \end{aligned}$$

□

Co dále? Výčet hodnot naznačuje pravidelnost a závěr, že obecný výsledek po b iteracích cyklu bude mít hodnotu

$$res \;=\; 7 + b(a + 2b + 8) = ab + 2b^2 + 8b + 7\,.$$

□

10.2 O „správnosti“ a dokazování programů

Jak se máme přesvědčit, že je daný program počítá „správně“?□

- Co třeba ladění programů? □
  Jelikož počet možných vstupních hodnot je (v principu) neohraničený, nelze otestovat všechna možná vstupní data.□

- Situace je zvláště komplikovaná v případě paralelních, randomizovaných, interaktivních a nekončících programů (operační systémy, systémy řízení provozu apod.). Takové systémy mají nedeterministické chování a opakované experimenty vedou k různým výsledkům.
  Nelze je tudíž ani rozumně ladit... □

- V některých případech je však třeba mít naprostou jistotu, že program funguje tak jak má, případně že splňuje základní bezpečnostní požadavky.□
  - Pro „malé“ algoritmy je možné podat přesný matematický důkaz□.
  - Narůstající složitosti programových systémů si pak vynucují vývoj jiných „spolehlivých“ formálních verifikačních metod. □

Mimochodem, co to vlastně znamená „počítat správně“?

Ukázka formálního důkazu algoritmu

Příklad 10.6. *Je dán následující symbolicky zapsaný algoritmus. Dokažte, že jeho výsledkem je „výměna“ vstupních hodnot a, b.*

Algoritmus 10.7.

```
input a, b;
a ← a+b;
b ← a-b;
a ← a-b;
output a, b;
```

Pro správný formální důkaz si musíme nejprve uvědomit, že je třeba symbolicky odlišit od sebe proměnné `a,b` od jejich daných vstupních hodnot, třeba h_a, h_b. Nyní v krocích algoritmu počítáme hodnoty proměnných:

- $a = h_a,\ b = h_b,$
- $a \leftarrow a + b = h_a + h_b, \quad b = h_b,$
- $a = h_a + h_b, \quad b \leftarrow a - b = h_a + h_b - h_b = \underline{h_a},$
- $a \leftarrow a - b = h_a + h_b - h_a = \underline{h_b}, \quad b = h_a,$

Tímto jsme s důkazem hotovi. $\square$

10.3 Jednoduché indukční dokazování

Pro dokazování algoritmů se jeví nejvhodněji matematická indukce, která je „jako stvořená“ pro formální uchopení opakovaných sekvencí v algoritmech. □

Příklad 10.8. *Dokažte, že násl. algoritmus navrátí výsledek* $ab + 2b^2 + 8b + 7$.

Algoritmus 10.9.

```
input a, b;
res ← 7;
foreach  i ← 1,2,...,b-1,b  do
    res ← res+a+2·b+8;
done
output res;
```

□

V prvé řadě si z důvodu formální přesnosti přeznačíme mez cyklu v algoritmu na `foreach i ← 1,2,...,c do ..` (kde $c = b$). Poté postupujeme přirozeně indukcí podle počtu c iterací (už nezávisle na vstupní hodnotě b); dokazujeme, že výsledek výpočtu algoritmu bude

$$res = (a + 2b + 8)c + 7 = ac + 2bc + 8c + 7\,.$$

Algoritmus .

```
input a, b;
res ← 7;
foreach  i ← 1,2,...,c-1,c  do
    res  ←  res+a+2·b+8;
done
output res;
```

Tvrzení: $res = ac + 2bc + 8c + 7$. □

Důkaz indukcí podle c:

* Pro $c = 0$ je výsledek správně $res = 0 + 7$. □
* Pokud dále předpokládáme platnost vztahu $res = ac + 2bc + 8c + 7$ po nějakých c iteracích cyklu `foreach`, tak následující iterace pro $i \leftarrow c + 1$ (jejíž průběh na samotné hodnotě i nezáleží) □změní hodnotu na

$$res \;\leftarrow\; res + a + 2b + 8 = ac + 2bc + 8c + 7 \;+\; a + 2b + 8 =$$
$$= a(c + 1) + 2b(c + 1) + 8(c + 1) + 7\,.$$

Důkaz indukcí je tím hotov. □

Příklad 10.10. *Zjistěte, kolik znaků* `'x'` *v závislosti na celočíselné hodnotě* n *vstupního parametru* `n` *vypíše následující algoritmus.*

Algoritmus 10.11.

```
foreach  i ← 1,2,3,...,n-1,n  do
    foreach  j ← 1,2,3,...,i-1,i  do
        vytiskni  'x';
    done
done
```
□

Nejprve si uvědomíme, že druhý (vnořený) cyklus vždy vytiskne celkem i znaků `'x'`. Proto iterací prvního cyklu (nejspíše) dostaneme postupně $1+2+\cdots+n$ znaků `'x'` na výstupu, což již víme (Příklad 2.8), že je celkem $\frac{1}{2}n(n+1)$.□ Budeme tedy dokazovat následující tvrzení:

Věta. Pro každé n Algoritmus 10.11 vypíše právě $\frac{1}{2}n(n+1)$ znaků `'x'`.

Algoritmus .

```
foreach  i ← 1,2,3,...,n-1,n  do
    foreach  j ← 1,2,3,...,i-1,i  do
        vytiskni  'x';
    done
done
```

Věta. Pro každé n Algoritmus 10.11 vypíše právě $\frac{1}{2}n(n+1)$ znaků `'x'`. □

Důkaz: Postupujeme indukcí podle n. Báze pro $n = 0$ je zřejmá, neprovede se ani jedna iterace cyklu a tudíž bude vytištěno 0 znaků `'x'`, což je správně. □

Nechť tedy tvrzení platí pro jakékoliv n_0 a položme $n = n_0 + 1$. Prvních n_0 iterací vnějšího cyklu podle indukčního předpokladu vypíše (ve vnitřním cyklu) celkem $\frac{1}{2}n_0(n_0+1)$ znaků `'x'`. □Pak již následuje jen jedna poslední iterace vnějšího cyklu s `i` ← `n=`n_0`+1` a v ní se vnitřní cyklus `j` ← `1,2,...,i=n` iteruje celkem $n = n_0 + 1$ -krát. □Celkem tedy bude vytištěn tento počet znaků `'x'`:

$$\frac{1}{2}n_0(n_0+1) + n_0 + 1 = \frac{1}{2}(n_0+1+1)(n_0+1) = \frac{1}{2}n(n+1)$$

Důkaz indukčního kroku je hotov. □

10.4 Rekurzivní algoritmy

* Rekurentní vztahy posloupností, stručně uvedené v Oddíle 3.4, mají svou přirozenou obdobu v rekurzivně zapsaných algoritmech. □
* Zjednodušeně řečeno to jsou algoritmy, které se v průběhu výpočtu odvolávají na výsledky sebe sama pro jiné (menší) vstupní hodnoty. □
* U takových algoritmů je zvláště důležité kontrolovat jejich správnost a také proveditelnost. □

Příklad 10.12. *Symbolický zápis jednoduchého rekurzivního algoritmu.*

Algoritmus . Rekurzivní funkce `factorial(x)`

```
if x < 1 then  t ← 1;
else  t ← x · factorial(x-1);
return t
```

Co je výsledkem výpočtu? □

Jednoduše řečeno, výsledkem je faktoriál vstupní přirozené hodnoty x, tj. hodnota $x! = x \cdot (x-1) \cdot \dots \cdot 2 \cdot 1$. □

Fibonacciho čísla

Pro další příklad rekurze se vrátíme k Oddílu 3.4, kde byla zmíněna známá Fibonacciho posloupnost $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$. Ve skutečnosti tuto posloupnost budeme uvažovat již od nultého členu, tj. jako $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$. □

Algoritmus 10.13. Rekurzivní výpočet funkce `fibonacci(x)`
Pro dané přirozené $x \geq 0$ vypočítáme x-té Fibonacciho číslo následovně:

```
if x < 2 then  t ← x;
else  t ←  fibonacci(x-1)+fibonacci(x-2);
return t
```

□

Správnost Algoritmu 10.13 je víceméně zřejmá z jeho přímé podoby s rekurentním vzorcem v definici Fibonacciho čísel. Zamyslete se však, jak je to s praktickou „proveditelností" takového algoritmu...

Co třeba $fibonacci(40)$ nebo $fibonacci(50)$?

Příklad 10.14. *Nerekurzivní algoritmus pro Fibonacciho čísla.*

Dokažte, že následující algoritmus pro každé přirozené n počítá tutéž hodnotu jako rekurentní funkce `fibonacci(n)` v Algoritmu 10.13.

Algoritmus .

```
input n;
b[0] ← 0;  b[1] ← 1;
foreach  i ← 2,3,...,n  do
    b[i]  ←  b[i-1]+b[i-2];
done
output b[n]
```

Indukcí budeme dokazovat, že po i-té iteraci cyklu algoritmu bude vždy platit $b[i] = fibonacci(i)$: Co se týče báze indukce, toto vyplývá z úvodního přiřazení.

* Pro libovolné $i \geq 2$ předpokládáme platnost indukčního předpokladu $b[j] = fibonacci(j)$ pro $j \in \{i-1, i-2\}$.
* V i-té iteraci cyklu nastane $b[i] \leftarrow b[i-1] + b[i-2] = fibonacci(i-1) + fibonacci(i-2) = fibonacci(i)$ podle definice. □