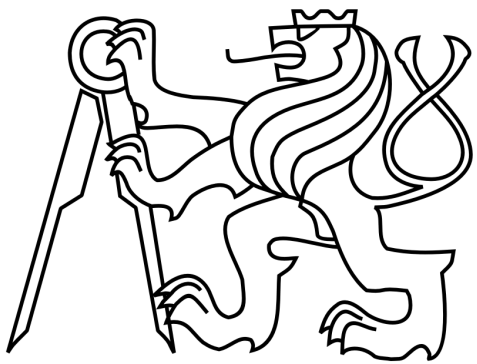
Formální Metody a Specifikace (LS 2011)
Přednáška 11:
Funkce, procedury, objekt-orientované programy

Stefan Ratschan

Katedra číslicového návrhu
Fakulta informačních technologií
Ceské vysoké učení technické v Praze

6. květen 2011



Příklad

$$r = \gcd(x, y) :\Leftrightarrow r|x \wedge r|y \wedge \neg\exists r' \;.\; r'|x \wedge r'|y \wedge r' > r$$

GCD(x,y)
@ $x, y \in \mathcal{N}, x \geq y$
if $y = 0$ **then**
    $r \leftarrow x$
else
    @ $y \neq 0 \wedge y \geq x \bmod y$
    $r \leftarrow \texttt{GCD}(y, x \bmod y)$
    @ $y \neq 0 \wedge r = \gcd(y, x \bmod y)$
@ $r = \gcd(x, y)$
return r

Ze samého řádku $r \leftarrow \texttt{GCD}(y, x \bmod y)$
    nevyplývají žádné nové ověřovací podmínky.

Volání funcí

$\ldots$
@ $\alpha \wedge \beta[v_1 \leftarrow t_1, \ldots, v_n \leftarrow t_n]$
$u \leftarrow f(t_1, \ldots, t_n)$
@ $\alpha \wedge \beta'[v_1 \leftarrow t_1, \ldots, v_n \leftarrow t_n, w \leftarrow u]$ // víme, nemusí se ověřit
$\ldots$

function $f(v_1, \ldots, v_n)$
@ β
$\ldots$
@ β'
return w

Během běhu program se nemusí ověřovat ekvivalentní aserce dvakrát.

f nesmí mít žádné vedlejší efekty (všechny proměnné lokální)

Volání funkce ve složitějším výrazu?

Volání funcí

Pokud α obsahuje u?

$\ldots$
@ $\alpha \wedge \beta[v_1 \leftarrow t_1, \ldots, v_n \leftarrow t_n]$
$u \leftarrow f(t_1, \ldots, t_n)$
@ $[\exists u \,.\, \alpha] \wedge \beta'[v_1 \leftarrow t_1, \ldots, v_n \leftarrow t_n, w \leftarrow u]$
$\ldots$

function $f(v_1, \ldots, v_n)$
@ β
$\ldots$
@ β'
return w

Procedury s vedlejšími efekty

...
@ $\alpha \wedge \beta[v_1 \leftarrow t_1, \ldots, v_n \leftarrow t_n]$
$p(t_1, \ldots, t_n)$
@ $[\exists x_1, \ldots, x_k \,.\, \alpha] \wedge \beta'[v_1 \leftarrow t_1, \ldots, v_n \leftarrow t_n]$
...

procedure $p(v_1, \ldots, v_n)$
@ β
...
@ β'
přičemž p smí změnit jen $x_1, \ldots, x_k$ (důležitá součást specifikace!)

Elegantnější: Místo změna globálních proměnných, vracení víc hodnot:

$$(x_1, \ldots, x_n) \leftarrow p(v_1, \ldots, v_n)$$

Terminace rekurze

GCD(x,y)
@ $x, y \in \mathcal{N}, x \geq y$
$v \leftarrow y$ // variant y
if $y = 0$ **then**
    $r \leftarrow x$
else
    @ $y \neq 0 \wedge y \geq x \bmod y \wedge x \bmod y < v \wedge x \bmod y \geq 0$
    $r \leftarrow \texttt{GCD}(y, x \bmod y)$
    @ $r = \gcd(y, x \bmod y)$
@ $r = \gcd(x, y)$
return r

Nová ověřovací podmínka (vlastně rozšíření staré podmínky):
$[x \geq y \wedge y \neq 0] \Rightarrow [\cdots \wedge x \bmod y < v \wedge x \bmod y \geq 0][v \leftarrow y] \quad \Leftrightarrow$
$[x \geq y \wedge y \neq 0] \Rightarrow [\cdots \wedge x \bmod y < y \wedge x \bmod y \geq 0]$

Binární vyhledávání

$\mathrm{search}(a, l, u, x)$
@ $a \in \mathcal{A}_n(\mathcal{N})$, $\mathrm{sorted}(a)$, $l, u \in \{1, \ldots, n\}$, $x \in \mathcal{N}$
if $u < l$ **then**
    $r \leftarrow 0$
else if $a[(l+u)/2] = x$ **then**
    @ $l \geq u$
    $r \leftarrow (l+u)/2$
else if $a[(l+u)/2] < x$ **then**
    @ $l \geq 0 \wedge \neg\exists i \,.\, [l \leq i \leq (l+u)/2 \wedge a[i] = x] \wedge \mathrm{sorted}(a)$
    $r \leftarrow \mathrm{search}(a, (l+u)/2+1, u, x)$
    @ $\neg\exists i \,.\, [l \leq i \leq (l+u)/2 \wedge a[i] = x] \wedge$
      $[r = 0 \Rightarrow [\neg\exists i \,.\, (l+u)/2+1 \leq x \leq u \wedge a[i] = x]] \wedge$
      $[r \neq 0 \Rightarrow [(l+u)/2+1 \leq r \leq u \wedge a[r] = x]]$
else
    $\ldots$
    $r \leftarrow \mathrm{search}(a, l, (l+u)/2, x)$
    @ $\ldots$
@ $r = 0 \Rightarrow [\neg\exists i \,.\, l \leq i \leq u \wedge a[i] = x] \wedge$
  $r \neq 0 \Rightarrow [l \leq r \leq u \wedge a[r] = x]$
return r

$$\mathrm{sorted}(a) :\Leftrightarrow \exists n \,.\, a \in \mathcal{A}_n \wedge \forall i \in \{1, \ldots, n-1\} \,.\, a[i] \leq a[i+1]$$

$$\mathrm{perm}(a, b) :\Leftrightarrow \exists n \,.\, a \in \mathcal{A}_n \wedge b \in \mathcal{A}_n \wedge \exists c \,.\, \mathrm{perm1n}(c, n) \wedge \forall i \in \{1, \ldots, n\} \,.\, b[c[i]] = a[i]$$

$$\mathrm{perm1n}(a, n) :\Leftrightarrow \exists n \,.\, a \in \mathcal{A}_n \wedge \forall i \in \{1, \ldots, n\}\ \exists j \in \{1, \ldots, n\} \,.\, a[j] = i \wedge \forall i, j \in \{1, \ldots n\} \,.\, i \neq j \Rightarrow a[i] \neq a[j]$$

$\mathrm{sort}(a)$
@ $a \in \mathcal{A}_n$
if $n = 1$ **then**
    $b \leftarrow a$
else
    $c \leftarrow \mathrm{sort}(a[1 \ldots (1+n)/2]$
    $d \leftarrow \mathrm{sort}(a[(1+n)/2+1 \ldots n])$
    $b \leftarrow \mathrm{merge}(c, d)$
    @ $\mathrm{sorted}(c) \wedge \mathrm{perm}(a[1 \ldots (1+n)/2], c) \wedge$
      $\mathrm{sorted}(d) \wedge \mathrm{perm}(a[(1+n)/2+1 \ldots n], d) \wedge$
      $\mathrm{perm}(c+d, b) \wedge \mathrm{sorted}(b)$
@ $b \in \mathcal{A}_n \wedge \mathrm{sorted}(b) \wedge \mathrm{perm}(a, b)$
return b

$\mathrm{merge}(a, b)$
@ $a \in \mathcal{A}_r, b \in \mathcal{A}_s, \mathrm{sorted}(a), \mathrm{sorted}(b)$
...
@ $\mathrm{perm}(a+b, r) \wedge \mathrm{sorted}(r)$
return r

Objektově orientované programování

class counter
constructor init(n)
method val(): $\mathcal{N}$
method dec()

Pozorování: Metody reprezentují funkce

init(): $\mathcal{N} \to$ counter
$val()$: counter $\to \mathcal{N}$
dec(): counter $\to$ counter

Objektově orientované programování

class counter
constructor init(n)
@ $n \in \mathcal{N}$
...
@ val() $= n$
method val(): $\mathcal{N}$
method dec()
$v \leftarrow val()$
...
@ $[v = 0 \Rightarrow \text{val}() = 0] \wedge [v > 0 \Rightarrow \text{val}() = v - 1]$

Používání:
$c \leftarrow$ init(2)
@ c.val() $= 2$
$v \leftarrow c$.val()
c.dec()
@ $[v = 0 \Rightarrow c.\text{val}() = 0] \wedge [v > 0 \Rightarrow c.\text{val}() = v - 1]$

Objektově orientované programování

Specifikace metod má být nezávislé od implementace
(viz. abstraktní datový typ)

Ale: Chceme zajišťovat vnitřní konzistenci implementace

Např.: Vnitřní proměnná x má vždy splňovat: $x \geq 0$.

Bylo by to aserce na začátku a na konci všech metod.

Některé objektově orientované programovací jazyky
mají explicitní podporu:

Eiffel: **invariant** $x \geq 0$,
ověřuje se vždy na začátku a na konce všech metod

Výhled

Chybí ještě hodně konstruktů:

- Dědičnost
- Ukazatele
- Souběžné procesy, vlákna
- ...

Příště:

- Velké přehled, souvislosti atd.
- Algoritmy pro automatické důkazy logických teorií.