12. přednáška – 2. část

Dokazování správnosti programů

10. a 11. prosince 2024

Dokazování správnosti algoritmu

- Správnost algoritmu
- Dokazování správnosti algoritmů je významnou oblastí výzkumu. Je v neustálém vývoji a hledá pro důkazy nejvhodnější postupy a metody a nástroje, včetně počítačové podpory.
- Základním principem dokazování je stanovení oborů hodnot proměnných a vztahů mezi proměnnými pro každý příkaz programu.
- Vztahy se vyjadřují tvrzeními (logickými výroky) vztahujícími se k jednotlivým místům programu a nezávislými na cestě, po níž se k danému místu v algoritmu dostaneme.

Pravidlo 1:

- Pro každý příkaz S algoritmu jsou nalezeny podmínky – tvrzení, které platí před a po provedení příkazu S. Tvrzení platné před příkazem se nazývá antecedence (angl. precondition) a po provedení příkazu se nazývá konsekvence (postcondition).

```
// Platí antecedence P
S
// Platí konsekvence Q
```

- Nechť příkaz S má jako antecedenci tvrzení P a konsekvenci Q, pak to zapisujeme notací $(S,P) \Rightarrow Q$.

Pravidlo 2:

Spojuje-li se před příkazem T několik větví algoritmu, pak konsekvence všech předcházejících příkazů S_i |(1<i<n) musí logicky implikovat antecedenci následujícího příkazu T.

```
case i of
  1: S₁ // platí konsekvence Q₁
  2: S₂ // platí konsekvence Q₂
  ...
  n: Sₙ // platí konsekvence Qₙ
end case
// platí Qᵢ⇒P | 1 ≤ i ≤ n kde P je antecedence příkazu T
T
```

Pravidlo 3:

Platí-li před podmíněným příkazem s podmínkou B tvrzení P, pak konsekvence příkazu za "then" je B and P a konsekvence příkazu za "else" je not B and P.

```
// P: −10 < x < 10
if x<0     // B
  then     // Qthen: −10 < x < 0
  else     // Qelse: 0 ≤ x < 10
```

Pravidlo 4:

Pro přiřazovací příkaz **v** ← **e** platí:
Nechť $P_{\mathbf{ev}}$ je antecedence příkazu **v** ← **e**.
Konsekvenci tohoto příkazu dostaneme tak, že každý výskyt výrazu **e** (expression) v antecedenci nahradíme proměnnou **v** (variable).

- Inverzní pravidlo říká, že je-li Q konsekvence přiřazovacího příkazu **v** ← **e**, pak jeho antecedenci získáme náhradou každého výskytu proměnné "**v**" v konsekvenci výrazem "**e**" .

Ilustrace pravidla 4:

```
// P1: y=x; d=2x-1 => d+2=2x+1
   d ← d+2
// Q1=P2: y=x; d=2x+1 => y+d=x+2x+1=3x+1
   y ← y+d
// Q2=P3: d=2x+1; y=3x+1 => d+2=2x+3
   d ← d+2
// Q3=P4: y=3x+1; d=2x+3 =>y+d=(3x+1)+2x+3=5x+4
   y ← y+d
// Q4: d=2x+3; y=5x+4
```

Příklad

Dokažte, že sekvence příkazů:

$$x \leftarrow x+y$$

$$y \leftarrow x-y$$

$$x \leftarrow x-y$$

Provede výměnu proměnných `x` a `y`

```
// P1:  x=X; y=Y => x+y=X+Y
    x ← x+y
// Q1=P2:  y=Y; x=X+Y => x-y=X+Y-Y=X
    y ← x-y
// Q2=P3:  x=X+Y; y=X => x-y=X+Y-X=Y
    x ← x-y
// Q3:  x=Y; y=X  Q.E.D.
```

Q.E.D. – quod erat demonstrandum (což mělo být dokázáno)

Pravidlo 5a pro cyklus while

- Nechť je dáno tvrzení P, které je invariantní (neměnné) vzhledem k příkazu S (provedení příkazu S nemá na tvrzení P žádný vliv; tvrzení je současně antecedencí i konsekvencí příkazu). Pak pro cyklus S' typu "while B do", jehož vnitřním příkazem je příkaz S, platí konsekvence ve tvaru P and not B.
- $(\mathrm{S}', \mathrm{P}) \Rightarrow \mathrm{P}\ and\ \overline{\mathrm{B}}$

```
// P
while B do    // příkaz S'
    S
end while
// konsekvence: P and not B
```

Ukázka pro dělení q = x div y

```
// x>0, y>0
q ← 0       // q=0 … podíl (quotient)
r ← x       // r=x … zbytek (remainder)
while r≥y do
    // P1: q*y+r=x; (r-y)≥0 =>(q+1)*y+(r-y)=x
    r ← r-y
    // Q1=P2: (q+1)*y+r=x; r≥0
    q ← q+1
    // Q2: q*y+r=x; r≥0
end while
// q*y+r=x; 0≤r<y … r<y ≡ not r≥y
```

Pravidlo 5b pro cyklus repeat:

Nechť pro příkaz S platí dva předpoklady:

$$(\mathrm{S}, \mathrm{P}) \Rightarrow \mathrm{Q}$$

$$(\mathrm{S}, \mathrm{Q}\ and\ \overline{\mathrm{B}}) \Rightarrow \mathrm{Q}$$

Pak cyklus S'' typu "repeat ... until B", který obsahuje uvnitř příkaz S a má antecedenci P, má konsekvenci Q and B

$$(\mathrm{S}'', \mathrm{P}) \Rightarrow \mathrm{Q}\ and\ \mathrm{B}$$

```
// P
  repeat
    S
  until B
// Q and B
```

S ohledem na skutečnost, že pro cyklus typu repeat-until se vyžaduje platnost obou předpokladů (P a také Q and not B), bývá tento cyklus častěji zdrojem chyb než cyklus typu while, i když jeho použití někdy vede ke kratšímu zápisu.

Ukázka pro násobení celých čísel pomocí sčítání a odčítání

```
// x>0; y>0
z ← 0    // z=0
u ← x    // u=x
repeat
    // P1: z+u*y=x*y; u>0 => z+y+(u-1)*y=x*y
    z ← z+y
    // Q1=P2: z+(u-1)*y=x*y; (u-1)≥0 )
    u ← u-1
    // Q2: z+u*y=x*y; u≥0 )
until u=0
// z+u*y=x*y; u=0 => z=x*y
```