Principi programskih jezikov

Rešitve

Pri dokazovanju pravilnosti si lahko pomagamo z naslednjimi splošnimi nasveti.

- Po stavku `x := N` vedno velja `{ x = N }`. Formalno to dokažemo tako: `{ N = N } x := N { x = N }`.
- Pravilo za prireditveni stavek je `{ P[x ↦ e] } x := e { P }`. Pri izpeljevanju najprej vse `x`-e v predpogoju izrazimo z `e`, nato pa `e`-je zamenjamo z `x`.
- V zančni invarianti se zančni pogoj in lokalne spremenljivke ne pojavljajo, saj mora invarianta veljati tudi po izstopu iz zanke. Ko invarianto »peljemo« čez telo zanke, se nam v njej lahko pojavijo lokalne spremenljivke. Teh se želimo znebiti, ko dosežemo konec zanke.
- Na koncu zanke se lahko izkaže, da je invarianta prešibka in iz nje ne moremo izpeljati končnega pogoja. V tem primeru lahko v invarianto dodamo še kak pogoj in ga ločeno peljemo čez zanko. To ne bo nikoli pokvarilo obstoječe izpeljave: če velja `A ⇒ C`, potem velja tudi `A ∧ B ⇒ C`.

Naloga 1

Dokažite parcialno in popolno pravilnost programa glede na dano specifikacijo.

```
{ x = m ∧ y = n }
x := x + y;
y := x - y;
x := x - y;
{ x = n ∧ y = m }
```

Rešitev

```
{ x = m ∧ y = n }
{ x + y = m + n ∧ y = n }
x := x + y;
{ x = m + n ∧ y = n }
{ x = m + n ∧ x - y = m }
y := x - y;
{ x = m + n ∧ y = m }
{ x - y = n ∧ y = m }
x := x - y;
{ x = n ∧ y = m }
```

Naloga 2

Dokažite parcialno in popolno pravilnost programa glede na dano specifikacijo.

```
{ }
if y < x then
  z := x;
  x := y;
  y := z
else
  skip
end
{ x ≤ y }
```

Rešitev

Rešujemo od spodaj navzgor (vendar na začetek `then` dodamo pogoj in na začetek `else` dodamo negacijo pogoja).

```
{ }
if y < x then
  { y < x }
  z := x;
  { y < x, z = x }
  { y < z }
  x := y;
  { x < z }
  y := z
  { x < y }
  { x ≤ y }
else
  { ¬(y < x) }
  { x ≤ y }
  skip
  { x ≤ y }
end
{ x ≤ y }
```

Naloga 3

Sestavite program `c`, ki zadošča specifikaciji

```
[ n ≥ 0 ]
c
[ s = 1 + 2 + ... + n ]
```

in dokažite njegovo pravilnost.

Prva rešitev

Ker smo dobri matematiki, znamo sešteti aritmetično vrsto:

```
1 + 2 + ... + n = n * (n + 1)/2
```

Torej lahko zapišemo program takole:

```
s := n * (n + 1) / 2
```

Druga rešitev

Program zapišemo z zanko `while`:

```
s := 0 ;
i := 1 ;
while i <= n do
  s := s + i ;
  i := i + 1
done
```

Dokažimo parcialno pravilnost:

```
{ n ≥ 0 }
s := 0 ;
{ s = 0 }
i := 1 ;
{ s = 0, i = 1 }
{ s = 1 + 2 + ... + (i - 1), i ≤ n + 1}
while i <= n do
  { i ≤ n, s = 1 + 2 + ... + (i - 1), i ≤ n + 1 }
  { i ≤ n, s = 1 + 2 + ... + (i - 1) }
  { i ≤ n, s + i = 1 + 2 + ... + (i - 1) + i }
  s := s + i ;
  { i ≤ n, s = 1 + 2 + ... + (i - 1) + i }
  { i + 1 ≤ n + 1, s = 1 + 2 + ... + (i + 1 - 1) }
  i := i + 1
  { i ≤ n + 1, s = 1 + 2 + ... + (i - 1) }
  { s = 1 + 2 + ... + (i - 1), i ≤ n + 1 }
done
{ i > n, s = 1 + 2 + ... + (i - 1), i ≤ n + 1, }
{ n < i ≤ n + 1, s = 1 + 2 + ... + (i - 1) }
{ i = n + 1, s = 1 + 2 + ... + (i - 1) }
{ s = 1 + 2 + ... + n }
```

Zanka se ustavi, ker se zmanjšuje nenegativna količina `n - i`. Tu je treba uporabiti `n ≥ 0`.

Pred `while` velja:

```
{ n - i = d, n - i ≥ - 1 }
```

Količina se zmanjša, ko se izvede telo zanke, in ohrani se spodnja meja:

```
{ i ≤ n, n - i = d, n - i ≥ - 1 }
s := s + i ;
{ i ≤ n, n - i = d, n - i ≥ - 1 }
{ i + 1 ≤ n + 1, n - (i + 1 - 1) = d, n - (i + 1 - 1) ≥ - 1 }
i := i + 1
{ i ≤ n + 1, n - (i - 1) = d, n - (i - 1) ≥ - 1 }
{ i ≤ n + 1, n - i = d - 1, n - i ≥ - 2 }
{ n - i ≥ -1, n - i = d - 1, n - i ≥ - 2 }
{ n - i ≥ -1, n - i = d - 1 }
```

Ker je `d - 1 < d`, se količina res zmanjšuje.

Naloga 4

Dokažite parcialno in popolno pravilnost programa glede na dano specifikacijo.

```
{ x ≥ 0 }
y := 0;
z := x;
while 1 < z - y do
  s := (y + z)/2;
  if s * s < x then
    y := s
  else
    z := s
  end
done
{ y² ≤ x ≤ (y+1)² }
```

Rešitev

Dokažimo parcialno pravilnost:

```
{ x ≥ 0 }
y := 0;
{ x ≥ 0, y = 0 }
z := x;
{ x ≥ 0, y = 0, z = x }
{ y² ≤ x ≤ z² }  # invarianta
while 1 < z - y do
  { y² ≤ x ≤ z², 1 < z - y }
  s := (y + z)/2;
  { y² ≤ x ≤ z², 1 < z - y, s = (y + z)/2 }
  if s * s < x then
    { y² ≤ x ≤ z², 1 < z - y, s = (y + z)/2, s² < x }  # y = 2s - z
    { (2s - z)² ≤ x ≤ z², 1 < z - (2s - z), s² < x }
    y := s
    { (2y - z)² ≤ x ≤ z², 1 < z - (2y - z), y² < x }
    { (2y - z)² ≤ x ≤ z², 1 < 2(z - y), y² < x }
  else
    { y² ≤ x ≤ z², 1 < z - y, s = (y + z)/2, s² ≥ x }  # z = 2s - y
    { y² ≤ x ≤ (2s - y)², 1 < (2s - y) - y, s² ≥ x }
    z := s
    { y² ≤ x ≤ (2z - y)², 1 < (2z - y) - y, z² ≥ x }
  end
  { y² ≤ x ≤ z² }
done
{ y² ≤ x ≤ z², 1 ≥ z - y }
{ y² ≤ x ≤ z², z ≤ y + 1 }
{ y² ≤ x ≤ (y+1)² }
```

Zanka se ustavi, ker se zmanjšuje nenegativna količina `z - y`. Pred `while` velja:

```
{ z - y = d, z - y > 0 }
```

Količina se zmanjša, ko se izvede telo zanke, in ohrani se spodnja meja:

```
{ z - y = d, z - y > 0 }
s := (y + z)/2;
{ z - y = d, z - y > 0, s = (y + z)/2 }
if s * s < x then
  { z - y = d, z - y > 0, s = (y + z)/2 }  # y = 2s - z
  { z - (2s - z) = d, z - (2s - z) > 0 }
  y := s
  { z - (2y - z) = d, z - (2y - z) > 0 }
  { 2(z - y) = d, 2(z - y) > 0 }
  { z - y = d/2, z - y > 0 }
else
  { z - y = d, z - y > 0, s = (y + z)/2 }  # z = 2s - y
  { 2s - y - y = d, 2s - y - y > 0 }
  z := s
  { 2(z - y) = d, 2(z - y) > 0 }
  { z - y = d/2, z - y > 0 }
end
{ z - y = d/2, z - y > 0 }
```

Ker je `d/2 < d` (za `d ≥ 1`), se količina `z - y` res zmanjšuje. Ker imamo celoštevilsko deljenje, se na vsaki iteraciji `z - y` zmanjša vsaj za ena.

Zadnja sprememba: petek, 30. marec 2018, 22:06

◄ Vaje: dokazovanje pravilnosti

Skok na...

Zapiski ►