Script generated by TTT

Title: Seidl: Functional Programming and Verification (02.11.2018)

Date: Fri Nov 02 08:30:10 CET 2018

Duration: 91:02 min

Pages: 28

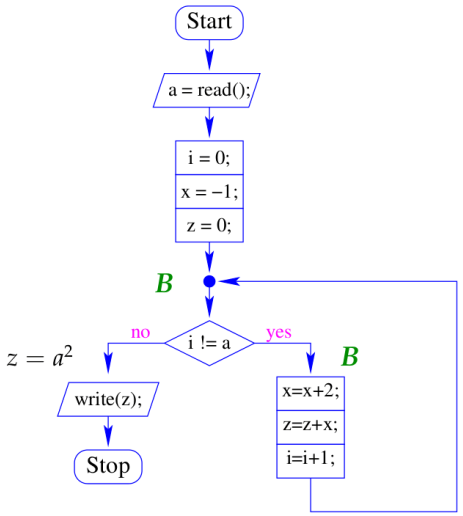
Example (cont.)

Assume $B \equiv z = i^2 \ \wedge \ x = 2i + 1$

Then we calculate:

$$
\begin{array}{lcll}
B_1 & \equiv & \mathbf{WP}[\![\texttt{i = i+1;}]\!](B) & \equiv \ z = (i+1)^2 \wedge x = 2(i+1) - 1 \\
 & & & \equiv \ z = (i+1)^2 \wedge x = 2i + 1 \\
B_2 & \equiv & \mathbf{WP}[\![\texttt{z = z+x;}]\!](B_1) & \equiv \ z + x = (i+1)^2 \wedge x = 2i + 1 \\
 & & & \equiv \ z = i^2 \wedge x = 2i + 1 \\
B_3 & \equiv & \mathbf{WP}[\![\texttt{x = x+2;}]\!](B_2) & \equiv \ z = i^2 \wedge x + 2 = 2i + 1 \\
 & & & \equiv \ z = i^2 \wedge x = 2i - 1 \\
 & & & \equiv \ B
\end{array}
$$

Example



Example (cont.)

Assume $B \equiv z = i^2 \ \wedge \ x = 2i - 1$

Then we calculate:

$$
\begin{array}{lcll}
B_1 & \equiv & \mathbf{WP}[\![\texttt{i = i+1;}]\!](B) & \equiv \ z = (i+1)^2 \wedge x = 2(i+1) - 1 \\
 & & & \equiv \ z = (i+1)^2 \wedge x = 2i + 1 \\
B_2 & \equiv & \mathbf{WP}[\![\texttt{z = z+x;}]\!](B_1) & \equiv \ z + x = (i+1)^2 \wedge x = 2i + 1 \\
 & & & \equiv \ z = i^2 \wedge x = 2i + 1 \\
B_3 & \equiv & \mathbf{WP}[\![\texttt{x = x+2;}]\!](B_2) & \equiv \ z = i^2 \wedge x + 2 = 2i + 1 \\
 & & & \equiv \ z = i^2 \wedge x = 2i - 1 \\
 & & & \equiv \ B
\end{array}
$$

Idea

- Make sure that each loop is executed only finitely often ...
- For each loop, identify an indicator value r, that has two properties

  (1) $r > 0$ whenever the loop is entered;

  (2) r is decreased during every iteration of the loop.
- Transform the program in a way that, alongside ordinary program execution, the indicator value r is computed.
- Verify that properties (1) and (2) hold!

Idea

- Make sure that each loop is executed only finitely often ...
- For each loop, identify an indicator value r, that has two properties

  (1) $r > 0$ whenever the loop is entered;

  (2) r is decreased during every iteration of the loop.
- Transform the program in a way that, alongside ordinary program execution, the indicator value r is computed.
- Verify that properties (1) and (2) hold!

We choose: $r = x + y$

Transformation

```
int a, b, x, y, r;
a = read(); b = read();
if (a < 0) x = -a; else x = a;
if (b < 0) y = -b; else y = b;
if (x == 0) write(y);
else if (y == 0) write(x);
     else { r = x+y;
        while (x != y) {
           if (y > x) y = y-x;
           else       x = x-y;
           r = x+y; }
        write(x);
}
```


At program points 1, 2 and 3, we assert:

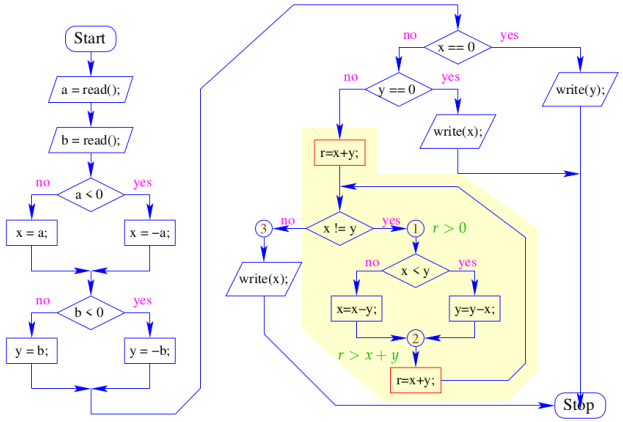
(1) $A \equiv x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y$

(2) $B \equiv x > 0 \wedge y > 0 \wedge r > x + y$

(3) **true**

Then we have:

$A \Rightarrow r > 0$ und $B \Rightarrow r > x + y$



At program points 1, 2 and 3, we assert:

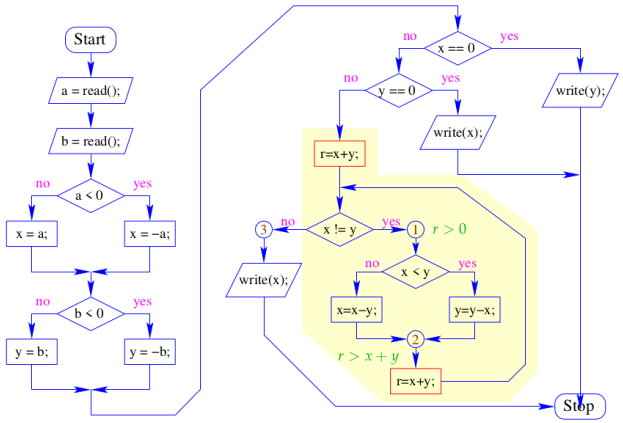
(1) $A \equiv x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y$

(2) $B \equiv x > 0 \wedge y > 0 \wedge r > x + y$

(3) **true**

Then we have:

$$A \Rightarrow r > 0 \quad \text{und} \quad B \Rightarrow r > x + y$$



88

At program points 1, 2 and 3, we assert:

(1) $A \equiv x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y$

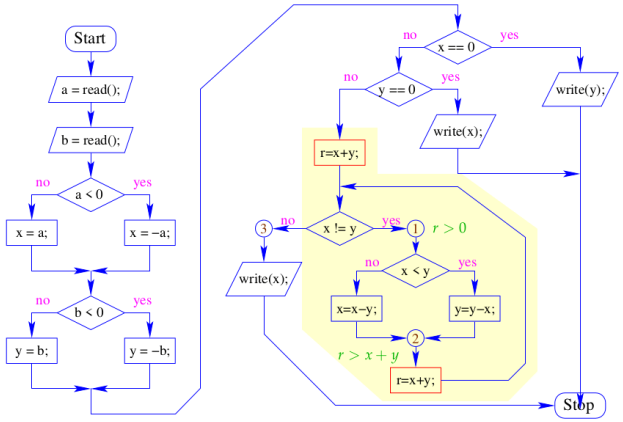
(2) $B \equiv x > 0 \wedge y > 0 \wedge r > x + y$

(3) **true**

Then we have:

$$A \Rightarrow r > 0 \quad \text{und} \quad B \Rightarrow r > x + y$$

89



At program points 1, 2 and 3, we assert:

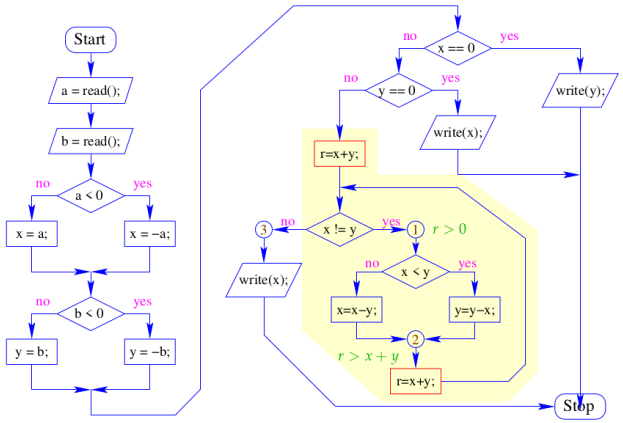
(1) $A \equiv x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y$

(2) $B \equiv x > 0 \wedge y > 0 \wedge r > x + y$

(3) **true**

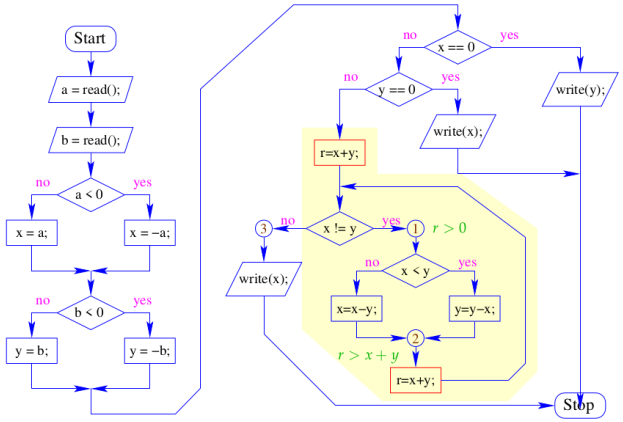
Then we have:

$$A \Rightarrow r > 0 \qquad \text{und} \qquad B \Rightarrow r > x + y$$



We verify:

$$\begin{aligned} \mathbf{WP}[\![\texttt{x != y}]\!](\mathbf{true}, A) &\equiv x = y \vee A \\ &\Leftarrow x > 0 \wedge y > 0 \wedge r = x + y \\ &\equiv C \end{aligned}$$



At program points 1, 2 and 3, we assert:

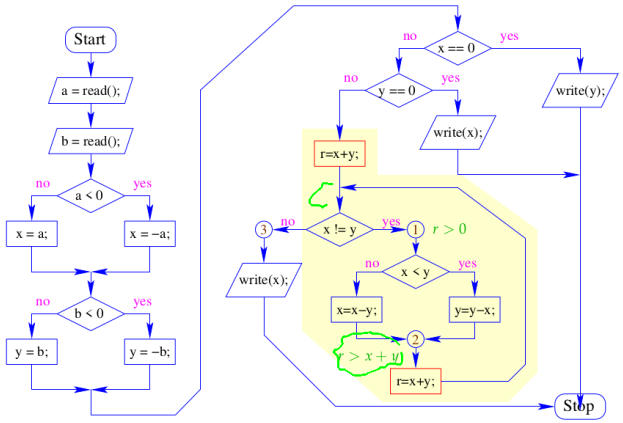
(1) $A \equiv x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y$

(2) $B \equiv x > 0 \wedge y > 0 \wedge r > x + y$

(3) **true**

Then we have:

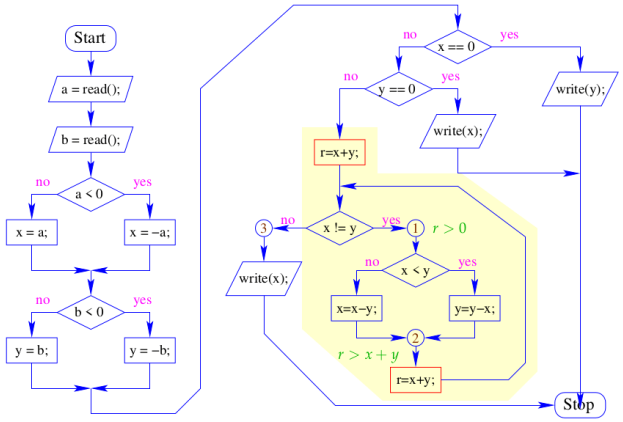
$$A \Rightarrow r > 0 \qquad \text{und} \qquad B \Rightarrow r > x + y$$



We verify:

$$\begin{aligned}
\mathbf{WP}[\![\texttt{x != y}]\!](\mathbf{true}, A) &\equiv x = y \vee A \\
&\Leftarrow x > 0 \wedge y > 0 \wedge r = x + y \\
&\equiv C \\
\mathbf{WP}[\![\texttt{r = x+y;}]\!](C) &\equiv x > 0 \wedge y > 0 \\
&\Leftarrow B \\
\mathbf{WP}[\![\texttt{x = x-y;}]\!](B) &\equiv x > y \wedge y > 0 \wedge r > x \\
\mathbf{WP}[\![\texttt{y = y-x;}]\!](B) &\equiv x > 0 \wedge y > x \wedge r > y \\
\mathbf{WP}[\![\texttt{y > x}]\!](\ldots,\ldots) &\equiv (x > y \wedge y > 0 \wedge r > x) \vee \\
&\quad\ (x > 0 \wedge y > x \wedge r > y) \\
&\Leftarrow x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y \\
&\equiv A
\end{aligned}$$

Orientation



General Method

- For every occurring loop `while (b) s` we introduce a fresh variable `r`.
- Then we transform the loop into:

```
r = e0;
while (b) {
   assert(r>0);
   s
   assert(r > e1);
   r = e1;
}
```

for suitable expressions `e0`, `e1`.

Idea

- Modularize the correctness proof in a way that sub-proofs for replicated program fragments can be reused.
- Consider statements of the form:

$$\{A\} \quad p \quad \{B\}$$

... this means:

If before the execution of program fragment p, assertion A holds and program execution terminates, then after execution of p assertion B holds.

Example

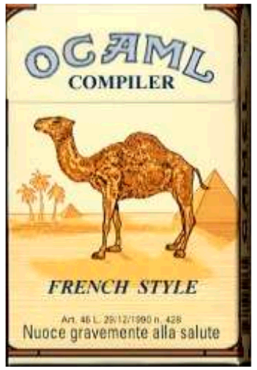
```
int x, m0, m1, t;

void main () {
   x = read();
   m0 = 1; m1 = 1;
   if (x > 1) f();
   write (m1);
}
```

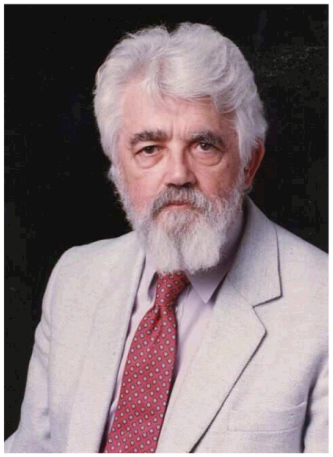
```
void f() {
   x = x-1;
   if (x>1) f();
   t = m1;
   m1 = m0+m1;
   m0 = t;
}
```

Summary

- Every further language construct requires dedicated verification techniques.
- How to deal with dynamic data-structures, objects, classes, inheritance ?
- How to deal with concurrency, reactivity ??
- Do the presented methods allow to prove everything $\Longrightarrow$ completeness ?
- In how far can verification be automated ?
- How much help must be provided by the programmer and/or the verifier ?

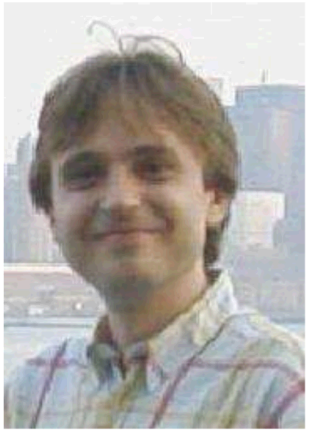
Functional Programming





John McCarthy, Stanford

Robin Milner, Edinburgh



Xavier Leroy, INRIA, Paris