Script **generated by TTT**

Title: Seidl: Functional Programming and Verification (26.10.2018)

Date: Fri Oct 26 08:22:10 CEST 2018

Duration: 85:19 min

Pages: 14

Example

```
public class GCD {
   public static void main (String[] args) {
   int x, y, a, b;
   a = read(); x = a;
   b = read(); y = b;
   while (x != y)
      if (x > y) x = x - y;
      else       y = y - x;

   assert(x == y);

   write(x);
   }  // End of definition of  main();
}     // End of definition of class GCD;
```


... in the GCD Program (1):

assignment: `x = x-y;`

post-condition: A

weakest pre-condition:

$$\begin{aligned} A[x-y/x] &\equiv gcd(a,b) = gcd(x-y,y) \\ &\equiv gcd(a,b) = gcd(x,y) \\ &\equiv A \end{aligned}$$

General Principle

- Every assignment transforms a post-condition B into a minimal assumption that must be valid before the execution so that B is valid after the execution.
- In case of an assignment `x = e;` the weakest pre-condition is given by

$$\mathbf{WP}[\![\texttt{x = e;}]\!]\ (B) \quad \equiv \quad B[e/x]$$

  This means: we simply substitute everywhere in B, x by e !!!
- An arbitrary pre-condition A for a statement s is valid, whenever

$$A \Rightarrow \mathbf{WP}[\![s]\!]\ (B)$$

  // A implies the weakest pre-condition for B.

... in the GCD Program (1):

assignment: `x = x-y;`

post-condition: A

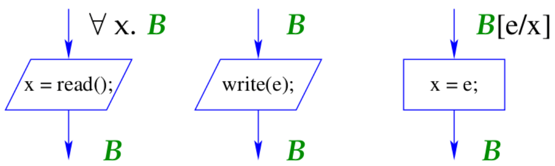
weakest pre-condition:

$$\begin{aligned} A[x-y/x] &\equiv gcd(a,b) = gcd(x-y,y) \\ &\equiv gcd(a,b) = gcd(x,y) \\ &\equiv A \end{aligned}$$

General Principle

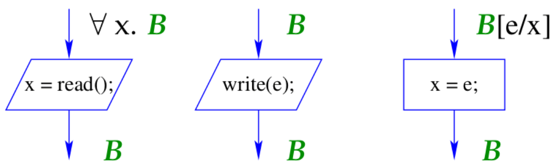
- Every assignment transforms a post-condition B into a minimal assumption that must be valid before the execution so that B is valid after the execution.

Wrap-up



$$\begin{aligned} \mathbf{WP}[\![\texttt{;}]\!](B) &\equiv B \\ \mathbf{WP}[\![\texttt{x = e;}]\!](B) &\equiv B[e/x] \\ \mathbf{WP}[\![\texttt{x = read();}]\!](B) &\equiv \forall x.\, B \\ \mathbf{WP}[\![\texttt{write(e);}]\!](B) &\equiv B \end{aligned}$$

Wrap-up



$$\begin{aligned} \mathbf{WP}[\![\texttt{;}]\!](B) &\equiv B \\ \mathbf{WP}[\![\texttt{x = e;}]\!](B) &\equiv B[e/x] \\ \mathbf{WP}[\![\texttt{x = read();}]\!](B) &\equiv \forall x.\, B \\ \mathbf{WP}[\![\texttt{write(e);}]\!](B) &\equiv B \end{aligned}$$

34

Discussion

- For all actions, the wrap-up provides the corresponding weakest pre-conditions for a post-condition B.
- An output statement does not change any variable. Therefore, the weakest pre-condition is B itself.
- An input statement `x=read();` modifies the variable `x` unpredictably.

  In order B to hold after the input, B must hold for every possible `x` before the input.

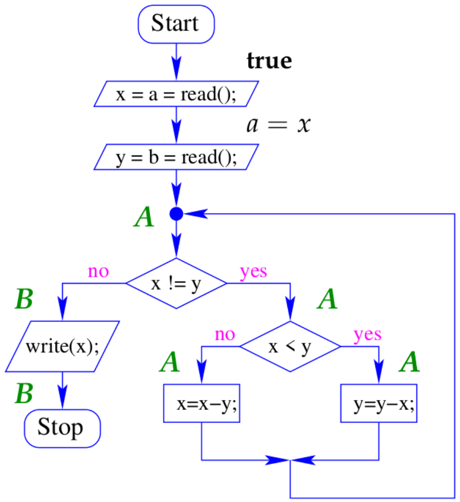
The argument for the assertion before the loop is analogous:

$$b \equiv y \neq x$$

$$\begin{aligned} \neg b \wedge B &\equiv B \\ b \wedge A &\equiv A \wedge x \neq y \end{aligned}$$

$\Longrightarrow$ $A \equiv (A \wedge x = y) \vee (A \wedge x \neq y)$ is the weakest pre-condition for the conditional branching.

Orientation



Summary of the Approach

- Annotate each program point with an assertion.
- Program start should receive annotation **true**.
- Verify for each statement s between two assertions A and B, that A implies the weakest pre-condition of s for B i.e.,

$$A \Rightarrow \mathbf{WP}[\![s]\!](B)$$

- Verify for each conditional branching with condition b, whether the assertion A before the condition implies the weakest pre-condition for the post-conditions B_0 and B_1 of the branching, i.e.,

$$A \Rightarrow \mathbf{WP}[\![b]\!]\ (B_0, B_1)$$

An annotation with the last two properties is called locally consistent.

Recap (2)

- An execution trace π traverses a path in the control-flow graph.
- It starts in a program point u_0 with an initial state σ_0 and leads to a program point u_m with a final state σ_m.
- Every step of the execution trace performs an action and (possibly) changes program point and state.

$\Longrightarrow$ The trace π can be represented as a sequence

$$(u_0, \sigma_0) s_1 (u_1, \sigma_1) \dots s_m (u_m, \sigma_m)$$

where s_i are elements of the control-flow graph, i.e., basic statements or conditions (guards) ...

Theorem

Let p be a MiniJava program, let π be an execution trace starting in program point u and leading to program point v.

Assumptions:

- The program points in p are annotated by assertions which are locally consistent.
- The program point u is annotated with A.
- The program point v is annotated with B.

Conclusion:

If the initial state of π satisfies the assertion A, then the final state satisfies the assertion B.

Remarks

- If the start point of the program is annotated with **true**, then every execution trace reaching program point v satisfies the assertion at v.
- In order to prove that an assertion A holds at a program point v, we require a locally consistent annotation satisfying:

  (1) The start point is annotated with **true**.

  (2) The assertion at v implies A.