

# **Program Analysis for Verification**

**Neelam Gupta**

**The University of Arizona  
Tucson, Arizona, USA**

# Program Analysis for Verification

Testing: verification by experimentation

Analysis: inspecting and reasoning about programs to understand its properties and capabilities.

## Informal Analysis techniques

- (i) *Code Walk-throughs* – select some test cases and simulate the execution of code or design specifications. State of execution recorded by hand.

## Guidelines

- Number of people 3 to 5
- Documentation available in advance
- Predefined duration of meeting
- Focus on discovery of errors not fixing them.
- Participants take role of designer, moderator, and secretary.
- Foster cooperation not evaluation of people.

## (ii) *Code inspections*

Goal: Examine code | design explicitly for presence of some common errors.

- Use of uninitialized variables
- Jumps into loops
- Nonterminating loops
- Off by one errors
- Array indices out of bound
- Comparison of equality for floating-point values

Meeting guidelines similar to code walk-throughs

- List beforehand types of errors the inspections are aimed at.
- Reports produced by analysis tools received in advance.

# Correctness Proofs: Formal Program Analysis

Merge two sorted arrays of  $n$  elements each.

```
i = 1; j=1; k=1;
While (k<=2*n) {
    if a(i)<b(j) then
        c(k) := a(i);
        i:= i+1;
    else
        c(k) := b(j);
        j := j+1;
    endif
    k := k+1;
} /* endwhile */
```

Correctness Proof: program's semantics implies its specification.

Example:

```

{true} : pre-condition
begin
  read(a); read(b);
  a+b=input1+input2 ← _____
  x:=a+b;
  x=input1+input2 ← _____
  write(x);
end
{output=input1+input2}: postcondition
```

Proof by backward substitution:

$\{x=5\} x:=x+1 \{x=6\}$

$\{x_{old}=5\} x_{new}:=x_{old}+1 \{x_{new}=6\}$

$x_{old}=5 \quad x_{old}+1 = 6 \quad \leftarrow$  backsubstitution

○

$\{\text{post-cond}(x \text{ replaced by } \text{exp})\} x := \text{exp} \{\text{post-cond}\}$

$\{z-43 > y+7\} x:=z-43 \{x > y+7\}$

Input and output statements treated as assignments  
from and to input file and output file respectively.

a := input1; b:= input2;

output:=x;

# Backward substitution for sequences of assignment statements

Ex.

$$\{a+b-43 > y+7\} z:=a+b \{z-43 > y+7\}$$
$$\{z-43 > y+7\} x:=z-43 \{x>y+7\}$$
$$\{a+b-43 > y+7\} z:=a+b; x:=z-43 \{x > y+7\}$$
$$\{F1\} S1 \{F2\} \text{ and } \{F2\} S2 \{F3\}$$
$$\Rightarrow \{F1\} S1; S2 \{F3\}$$

Hoare's Notation for a proof rule for sequencing statements.

$\{F1\} S1 \{F2\} , \{F2\} S2 \{F3\}$

-----

$\{F1\} S1; S2 \{F3\}$

Claim1, Claim2

-----

Claim3

If claim1 and claim2 have been proven, one can deduce the claim3 to be proven.

# Proof rule for conditional statements

## Example

{true}

if  $x \geq y$  then

$\text{max} := x;$

else

$\text{max} := y;$

endif

{( $\text{max} = x$  or  $\text{max} = y$ ) and ( $\text{max} \geq x$  and  $\text{max} \geq y$ )}.

$\longrightarrow \{(x = x \text{ or } x = y) \text{ and } (x \geq x \text{ and } x \geq y)\}$

More generally

let the conditional statement be:

if cond then S1; else S2; endif;

*Proof Rule for Conditional Statements*

$\{\text{Pre and cond}\}S1\{\text{Post}\}, \{\text{Pre and not cond}\}S2\{\text{Post}\}$

---

$\{\text{Pre}\} \text{if cond then } S1; \text{ else } S2; \text{ endif; } \{\text{Post}\}$

*Proof Rule (while loops)*

Let I be an assertion

$\{I \text{ and cond}\} S \{I\}$

---

$\{I\} \text{ while cond loop } S; \text{ endloop; } \{I \text{ and not cond}\}$

Example.

$\{x \geq 0\}$

while  $x > 0$  loop

$x := x - 1$

endloop

$\{x = 0\}$

$\{x \geq 0\}$  is a loop invariant

$\{I \text{ and cond}\} S \{I\}$

$\{x \geq 0 \text{ and } x > 0\} x = x - 1 \{x \geq 0\}$  -----(1)

Backward substitution into the postcondition gives  $x - 1 \geq 0$ ,  
i.e.,  $x \geq 1$ .

$\{x \geq 0 \text{ and } x > 0\} \Rightarrow x \geq 1$

Therefore (1) is true

Therefore, by proof rule for loops,  $\{I \text{ and not cond}\} = \{x = 0\}$  is the  
postcondition

Thus, the proof rule for loops allows derivation of a postcondition  
for a while loop if it terminates.

Another Example:

{ $x > y$ }

while  $x \neq 0$  loop

$x := x - 2;$

$y := y - 2;$

endloop

{ $x > y$  and  $x = 0$ }

Proof rule for loops can be used to show the correctness of the above code if it terminates.

However, these are only partial correctness proofs.

Total correctness proof

= partial correctness proof

+

termination proof

## Another Example:

{input1 > 0 and input2 > 0}

begin

  read(x); read(y);

  div:=0;

  while x>= y loop

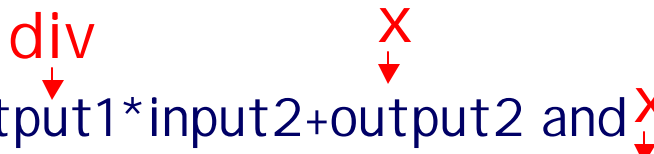
    div:=div+1;

    x:=x-y;

  endloop

  write(div); write(x);

end

{input1=output1\*input2+output2 and  $0 \leq \text{output2} < \text{input2}$ }.  


Integer Division Algorithm

The goal is to prove the correctness of the program: that is if the program begins in a state satisfying its pre-condition, then if it terminates, it will terminate in a state satisfying its post-condition.

We begin with applying back-substitution on the post-condition using  $\text{output2} := x$ ; and  $\text{output1} = \text{div}$ .

Thus we obtain the following post-condition:

$\{\text{input1} = \text{div} * \text{input2} + x \text{ and } 0 \leq x < \text{input2}\}$

This post-condition must be true at the termination of while loop in order for the post-condition of the program to be true.

Now, in order to prove that the above post-condition will be true at the termination of the loop, we need to find a suitable loop invariant.

Loop Invariants:  $z+w=2$ ;  $\text{div} \geq 0$ , but not useful.

We need a loop invariant that along with the negation of loop condition implies the truth of post-condition  $\{\text{input1} = \text{div} * \text{input2} + x \text{ and } 0 \leq x < \text{input2}\}$

If we can show that  $I1: \{y = \text{input2}\}$ ,

$I2: \{\text{input1} = \text{div} * y + x\}$  and  $I3: \{0 \leq x\}$  are all loop invariants then

$I = \{y = \text{input2} \text{ and } \text{input1} = \text{div} * y + x \text{ and } 0 \leq x\}$  will be loop invariant.

$\{I \text{ and } \text{not cond}\}$  will be then the desired postcondition.

Applying back-substitution to  $I1$  does not change  $I1$ . Thus  $I1$  is loop invariant.

$I_2: \{\text{input1} = \text{div} * y + x\}$

$I_2: \{\text{input1} = (\text{div}+1)*y + (x-y)\} = \{\text{input1} = \text{div} * y + x\}$

Therefore  $I_2$  is loop invariant

Now it remains to prove that  $I_3: \{x \geq 0\}$  is also loop invariant.

Applying back-substitution to  $\{x \geq 0\}$ , we obtain

$\{x-y \geq 0\}$  i.e.,  $x \geq y$ .

Now  $\{x \geq 0 \text{ and } x \geq y\} \Rightarrow \{x \geq y\}$

Therefore  $I_3$  is loop invariant.

Thus  $I$  is loop invariant.

Therefore, the loop invariant  $I$  should be true before the execution of the while loop for its post-condition to be true.

Now we continue to prove the correctness of the given program. Continuing with back-substitution in the invariant  $I$ , using  $\text{div}:=0$ , we get  $I = \{\text{y}=\text{input2} \textbf{ and } \text{input1}=\text{x} \textbf{ and } 0 \leq \text{x}\}$ .

Now applying back-substitution using  $\text{y}:=\text{input2}$ ,

we obtain,  $I : \{\text{input2}=\text{input2} \textbf{ and } \text{input1}=\text{x} \textbf{ and } 0 \leq \text{x}\}$ .

Further applying back-substitution using  $\text{x}:=\text{input1}$ ,

we obtain,  $I = \{\text{input2}=\text{input2} \textbf{ and } \text{input1}=\text{input1} \textbf{ and } 0 \leq \text{input1}\}$ .

Therefore,

$\{\text{input2}=\text{input2} \textbf{ and } \text{input1}=\text{input1} \textbf{ and } 0 \leq \text{input1}\} = \{0 \leq \text{input1}\}$ , must be true at the beginning of the program execution for the post condition of the program to be true. Clearly the pre-conditions of the program imply that  $\{0 \leq \text{input1}\}$  will be true at the beginning of the execution of the program. Therefore, the given program is correct.