Program Correctness

Program specification

Input: description of valid input to the program
Output: description of output of the program

Two parts to showing program correctness:

1. **Partial correctness:** For all valid input, if the program terminates on the input, then the program produces the correct output for that input. $(\forall x)P(x) \Rightarrow Q(x)$
2. **Termination:** The program terminates on every valid input. $(\forall x)\ P(x)$

Example: powering problem

Input: $a \in \mathbf{Z}$, $b \in \mathbf{Z}^+$ integer a, positive integer b.
Output: $a^b = \underbrace{a \cdot a \cdot \ldots \cdot a}_{b \text{ times}}$ (i.e., *a* multiplied *b* times)

Algorithm:

procedure *it_power*(*a*, *b*) ← for assignment (sybook use :=)
(1) $r \leftarrow a$ = for equality
(2) $c \leftarrow 1$
(3) while $c < b$ do
(4) $\quad r \leftarrow r \cdot a$
(5) $\quad c \leftarrow c + 1$
(6) end
(7) return *r*

Loop invariant: statement that holds each time the loop condition is about to be tested. e.g $c<b$ on line 3.

Claim: The following are **loop invariants** for the *it_power* algorithm:

(a) $r = a^c$ (b) $c \in \mathbf{Z}$ (c) $c \le b$

Suppose we have proved this claim. **Prove the correctness of the *it_power* algorithm:**

Partial correctness:

Assume we have valid input and our program terminate on this input.
Since program returning r. we need to show when the loop exists, $r=a^b$
When loop exists, it does so because $c<b$ evaluate to false,
This means $c \geq b$ — $c=b$
Loop invariant (c) says $c \leq b$
Loop invariant a) says $r = a^c = a^b$
The program then return r with the correct power

Termination:

Show if input is valid, the [illegible] will terminate

Assume input is valid, i.e. a, b

Only reason program won't terminate that is if we get an infinite loop

Loop exists when $c \leq b$ (line 3)

c start at line 2.

c increase by 1 in each iteration of the loop.

b doesn't change

Since b is finite, positive, & nonchanging, at some point $c > b$ become true.

So, loop can not be infinite.

Now **prove the claim using induction**:

***P*(*n*):** Loop invariants (a), (b), and (c) hold when the loop condition (on line 3) is tested for the *n*-th time.

Base case: Show P(1) holds, i.e., the loop invariants hold when $c < b$ is tested for the first time.

At point, $r = a$, $c = 1$ and $b \in \mathbb{Z}^+$ (since we assumed valid input).

So (a) $r = a = a^1 = a^c$

(b) $c \in \mathbb{Z}$ c) $c \leq b$ since $b \geq 1$

Inductive step: Show $P(k) \Rightarrow P(k+1)$

Induction hypothesis: The loop invariants hold the *k*-th time the loop is tested. i.e $P(k)$

Show loop invariants hold the $(k+1) \longrightarrow$ the loop is tested.

Let r_i, c_i be the values of *r*, *c* when the loop is about to be tested for the *i*-th time.

Suppose loop is tested for $(k+1)^{th}$ time.

This means loop had to be tested k times previously.

So, the k^{th} time the loop was tested, by IH,

$r_k = a^{c_k}$, $c_k \in \mathbb{Z}$, $c_k \leq b$.

Since we did execute the loop again, $c_k < b$.

During k^{th} iteration of the loop

$r_{k+1} \leftarrow r_k \cdot a$

$c_{k+1} \leftarrow c_k + 1$

Then, $r_{k+1} = r_k \cdot a = a^{c_k} \cdot a = a^{c_k+1} = a^{c_{k+1}}$ (a) holds.

Since 1, $c_k \in \mathbb{Z}$, $c_{k+1} = c_k + 1 \in \mathbb{Z}$.

Since $c_k < b$ and $c_k, b \in \mathbb{Z}$, $c_k \leq b-1$, $c_{k+1} \leq b$.

$\therefore$ by induction, $P(n)$ holds $\forall n \in \mathbb{Z}^+$

Program Correctness (continued)

Example: GCD

Definition: The *greatest common divisor* of integers *a* and *b*, gcd(*a*, *b*), is the largest integer *d* such that *d* divides both *a* and *b*. Note that:

gcd (15, 24) = 3.

- gcd(0, 0) is undefined
- gcd(*a*, 0) = *a* if *a* > 0
- gcd(0, *b*) = *b* if *b* > 0

Lemma: Let *a*, *b* ∈ **N** with at least one of *a*, *b* nonzero. Then the following three properties hold:

i. If *a* = *b*, then gcd(*a*, *b*) = *a* = *b*
ii. If *a* < *b*, then gcd(*a*, *b*) = gcd(*a*, *b* - *a*)
iii. If *a* > *b*, then gcd(*a*, *b*) = gcd(*a* - *b*, *b*)

Program specification: **Input:** *a*, *b* ∈ $\mathbf{Z}^+$ ⇐ a≠0, b≠0
Output: gcd(*a*, *b*)

Algorithm:

procedure *gcd*(*a*, *b*)
(1) (*x*, *y*) ← (*a*, *b*)
(2) while *x* ≠ *y* do
(3) if *x* < *y* then *y* ← *y* - *x*
(4) else *x* ← *x* - *y*
(5) end
(6) return *x*

Trace gcd (26, 18)

x	y
26	18
8	10
6	2
4	2
2	2

gcd (26, 18)
= gcd (8, 18)

Loop invariants

Invariants: After *n* iterations of the loop on line 2

(a) gcd(*a*, *b*) = gcd(*x*, *y*) (b) *x* > 0 (c) *y* > 0

<u>Proof by induction on # of iterations of the loop</u> :

Let x_i and y_i be the values of *x* and *y*, respectively, after *i* iterations of the loop.

P(*n*) : After *n* iterations of the loop, *gcd*(*a*, *b*) = *gcd*(*x*, *y*), *x* > 0, and *y* > 0   i.e $gcd(a,b) = gcd(x_n, y_n)$. $x_n > 0$, $y_n > 0$

Base case : P(0) holds $x_0 = a$ $y_0 = b$. and $a, b \in Z^+$

So $gcd(a, b) = gcd(x_0, y_0)$

Inductive step : P(*k*) ⇒ P(*k*+1)

Induction hypothesis : $gcd(x_k, y_k) = gcd(a,b)$, $x_k > 0$, $y_k > 0$ (24)

Case $x_k = y_k$. Then there won't be a $(k+1)^{st}$ iteration

So assume $x_k \neq y_k$.

Case $x_k < y_k$:

$x_{k+1} = x_k$ $y_{k+1} = y_k - x_k$

So $gcd(x_{k+1}, y_{k+1}) = gcd(x_k, y_k - x_k) = gcd(x_k, y_k)$ by lemma (2)

$= gcd(a, b)$ by IH.

$x_{k+1} = x_k > 0$ by IH.

Since $x_k < y_k$ $y_k - x_k = y_{k+1} > 0$

Case $y_k < x_k$: $x_{k+1} = x_k - y_k$ $y_{k+1} = y_k$.

So $gcd(x_{k+1}, y_{k+1}) = gcd(x_k - y_k, y_k) = gcd(x_k, y_k) = gcd(a,b)$ by IH.

Thus, in every case, $gcd(x_{k+1}, y_{k+1}) = gcd(a,b)$ $x_{k+1} > 0, y_{k+1} > 0$

Therefore, by induction, P(*n*) holds for all natural numbers *n*.

so $P(k+1)$ holds.

Prove the correctness of the *gcd* algorithm

Partial correctness : Prove for all valid input, if *gcd* algorithm terminates, then it produces the correct result.

Proof : Assume we have valid input and the algorithm terminates.

when loop in line(2) terminate, it is because $x = y$

At that point, the gcd of x, y is x which is returned on line (6).

Invariant (a) says $gcd(a,b)$ is the same as $gcd(x,y)$.

so the value we return is $gcd(a,b)$ □

Termination : Prove for all valid input the *gcd* algorithm terminates

Proof : Assume we have valid input.

Consider the quantity *x* + *y*. We'll first show that *x* + *y* decreases by at least 1 on each iteration.

when $x = y$, loop ends & program terminates.

suppose $x \neq y$ after k iters of loop, i.e. $x_k \neq y_k$.

Then either $x_{k+1} = x_k$ and $y_{k+1} = y_k - x_k$ (if $x_k < y_k$).

or $x_{k+1} = x_k - y_k$ and $y_{k+1} = y_k$ (if $y_k < x_k$).

Invariants b & c say $x_k, y_k >$, i.e. $x_k, y_k \geq 1$. so either $y_{k+1} \leq y_{k-1}$ or $x_{k+1} \leq x_{k-}$.

Then $x_{k+1} + y_{k+1} \leq x_k + y_{k-1}$ in either case.

Consider what happens after *m* iterations.

$x_m + y_m \leq x_{m-1} + y_{m-1} - 1 \leq x_{m-2} + y_{m-2} - 2 \dots \leq x_0 + y_0 - m$.

Initially $x_0 = a$, $y_0 = b$, where $a, b \in Z^+$.

Consider when *m* = *a* + *b* - 2

Invariant b & c say $x_m, y_m > 0$ so $x_m + y_m \geq 2$.

$2 \leq x_m + y_m \leq x_0 - y_0 - m = a + b - (a+b-2) = 2$.

i.e. $2 \leq x_m + y_m \leq 2$ so $x_m + y_m = 2$. thus $x_m = y_m = 1$.

So algorithm terminates after at most $m = a+b-2$. iterations.

(i.e, we found an upper bound on # iterations of while loop.

∴ program must terminate.