Program Correctness

Def: A program is **correct** if it produces the correct output for every possible input.

↳ by definition, we cannot prove a program w/ infinitely many valid inputs (e.g. the Euclidean Algorithm) is correct by simply trying test inputs.

↳ Even if there are only finitely many valid inputs, it may still be useful to prove the program correct (e.g. when the answers are not already known).

Proving Programs Correct

I. $T \Rightarrow C$ "Partial Correctness"

If the program terminates
Then the correct answer is produced

II. $V \Rightarrow T$ "Termination"

If the program receives valid input
Then the program terminates

Def: The **initial assertion** is a list of properties specifying what constitutes valid input.

Def: The **final assertion** is a list of properties that the output of the program needs to have (this essentially determines what it means for the program to be correct)

Notation: Given $p :=$ initial assertion

$q :=$ final assertion

& $S :=$ A portion of code

we write $p\{S\}q$ to stand for "S is partially correct with respect to the initial assertion p & final assertion q."

Rules of Inference

program

$S = \{S_1; S_2\}$

the program S is made up of two program segments S_1 followed by S_2

$$\begin{array}{l} p\{S_1\}q \\ q\{S_2\}r \\ \hline \therefore p\{S\}r \end{array}$$

"Composition rule"

↳ used to break up the burden of proving S is correct into two simpler tasks.

If condition Then
S

$$\begin{array}{l} (p \wedge \text{condition})\{S\}q \\ (p \wedge \neg\text{condition}) \rightarrow q \\ \hline \therefore p\{\text{If condition Then } S\}q \end{array}$$

"Conditional Statements"

If condition Then
  S_1
Else
  S_2

$$\frac{(p \wedge \text{condition})\{S_1\}q \qquad (p \wedge \neg\text{condition})\{S_2\}q}{\therefore p\{\text{if condition then } S_1 \text{ else } S_2\}q}$$

"if else Statements"

"Loop Invariants"

while condition
  S

$$\frac{(p \wedge \text{condition})\{S\}p}{\therefore p\{\text{while condition } S\}(\neg\text{condition} \wedge p)}$$

Note: to use these rules of inference, we must always prove partial correctness of some code segment

Some Basic Examples

initial assertion

factorial($n \geq 0$)
if $n=0$ then return 1
else return $n \cdot$factorial($n-1$)
{output is $n!$}

final assertion

we take the entire program as S

I We prove $p\{S\}q$

proof: We induct on n

base case: $n=0$ i.e. we prove

$$(\varnothing \wedge \underbrace{n=0}_{\text{condition}})\{\text{If } n=0 \text{ Then return } 1\}\text{output is } 0!$$

but this is clear since $0! = 1$

Inductive Hypothesis:

$n = k \{S\}$ output is $k!$

Induction Step:

Suppose $n = k+1$ then $(k+1)\text{Factorial}(k)$ is returned which equals $(k+1)\cdot\underbrace{k!}_{\text{by inductive hypothesis}} = (k+1)!$

This completes the proof of partial correctness $\square$

Note: The above proof by induction of the statement

$n \geq 0 \{S\} n!$ is basically the same as the proof using the "if-else" rule of inference

$$\begin{array}{ll} \text{base case} \rightarrow & (n \geq 0 \wedge n = 0)\{\text{return } 1\}\, n! \\ \text{inductive step} \rightarrow & (n \geq 0 \wedge n > 0)\{\text{return } n\cdot\text{factorial}(n-1)\}\, n! \\ \hline & \therefore\ n \geq 0 \{S\}\, n! \leftarrow \text{partial correctness} \end{array}$$

II $n \geq 0 \Rightarrow S$ terminates

proof: we induct on n

base case: $n = 0$ in this case, the code immediately returns 1

Inductive Hypothesis: S terminates with input k

Inductive Step: when S is run with input $k+1$ the program immediately returns $(k+1)\underbrace{\text{factorial}(k)}_{\text{terminates by IH}}$

Thus we have established "Termination" $\square$

$\Longrightarrow$ We have proven the recursive factorial function above is correct.

Exercise: Prove correctness of the following program

```
power(a ∈ ℤ-{0} ; n ≥ 0)
If n=0 Then return 1
Else return a·power(a, n-1)
{output is a^n}
```

Iterative Factorial Correctness Proof

S_1 $\left\{\begin{array}{l} i := 1 \\ \text{factorial} := 1 \end{array}\right.$

while $i < n$

S_2 $\left\{\begin{array}{l} i := i+1 \\ \text{factorial} := \text{factorial} \cdot i \end{array}\right.$

return factorial

Loop Invariant

$p :=$ "$\text{factorial} = i!$ & $i \leq n$"

proof: p & $i < n$ implies

$i_{new} = i+1 \leq n$ (= half of p)

$$\text{factorial}_{new} = \text{factorial} \cdot (i_{new}) = \underbrace{i!(i+1) = (i+1)!}_{\text{other half of } p}$$ $\square$

$$\frac{(p \wedge i<n)\{S_2\}p}{\therefore\ p\{\text{while } i<n\ S_2\}(\neg(i<n) \wedge p)}$$

Clearly, $(n \geq 1)\{S_1\}p$

So we have $(n \geq 1)\{S_1\}p$

$$\frac{(p \wedge i<n)\{S_2\}p}{\therefore\ p\{\text{while } i<n\ S_2\}(\neg(i<n) \wedge p)}$$

which shows $(n \geq 1)\{S_1; S_2\}\ factorial = n!$

Hence we have established partial correctness (I)

For (II) we note that $i := 1$ in S_1 so after $n-1$ iterations of the while loop, i will be given the value n $\Rightarrow$ while loop does not execute again. At this point the program returns

$\Rightarrow$ the program always terminates.

The Euclidean Algorithm

Recall: (lesson 15)

Def: the **Greatest Common Divisor** of a, b is a positive integer d with the following properties

final assertions
$$\begin{cases} \text{(i) } d \mid a & (\text{i.e. } d \in Div(a)) \\ \text{(ii) } d \mid b & (\text{i.e. } d \in Div(b)) \\ \text{(iii) } \forall e \text{ satisfying (i) \& (ii)} \\ \quad d \geq e & \end{cases}$$

(i.e. $d \in \max\{Div(a) \cap Div(b)\}$)

Claim: $\overbrace{a \in \mathbb{N},\ b \in \mathbb{N}-\{0\}}^{\text{initial assertion}} \Rightarrow$ gcd(a,b) exists & is unique

Proof:

Existence: $1 \mid x \ \forall x \in \mathbb{N}$

$\Rightarrow 1 \in Div(a) \cap Div(b) \neq \emptyset$

Note: Every finite subset of the naturals has a largest element.

Uniqueness: $\mathbb{N}$ is totally ordered

(in particular, if $a \leq b$ & $b \leq a$ then $a = b$)

Suppose c, d both satisfy (i), (ii) & (iii)

c satisfies (i) & (ii) $\Rightarrow c \leq d$ because d " (iii)

d satisfies (i) & (ii) $\Rightarrow d \leq c$ because c " (iii)

it follows that $c = d$ (any two things with the properties of a gcd must be equal so there is a unique such thing when one exists) $\square$

Lemma 0 $a = 0 \Rightarrow \gcd(a,b) = b$

Proof: $\forall x,\ x \mid 0 \Rightarrow b \mid 0$ & $b = \max\{Div(b)\}$ $\square$

$(Div(b) = \mathbb{N})$ $\quad (Div(a) \cap Div(b) = Div(a))$

Lemma 1: $b = qa + r,\ r < b \Rightarrow \gcd(a,b) = \gcd(a,r)$

Proof: Lesson 15 – Idea: $Div(a) \cap Div(b) = Div(a) \cap Div(r)$ $\square$

Recursive Euclidean Algorithm

This is the "Repeated Division Algorithm" version from lesson 15

$\gcd(a,b)$ $\quad a,b \in \mathbb{N}$ & $a < b$

If $a = 0$ Then return b

Else return $\gcd(a, \underbrace{b \bmod a}_{\text{division algorithm}})$

{output satisfies (i), (ii) & (iii)}

Claim: The above program is correct

"Inductive" Proof: $p \, S \, q$

Case: $a=0$

case: $a>0$

Since $b \bmod a < b$ one input is eventually zero reducing the problem to the above case

Lemma 0

Lemma 1

If Then Else proof:

If $a=0$ then S_1 q

Else S_2 q

Iterative Euclidean Algorithm (division is repeated subtraction)

$x := a$
$y := b$

while $x \neq y$ do

S

if $x<y$ then
$y = y - x$
else
$x = x - y$

Repeated Division Algorithm

We repeatedly subtract the smaller of x & y from the larger. It will often happen that the larger number flips between x & y (possibly many times) but we don't stop until $x=y$ at which point their shared value is $\gcd(a,b)$

P - loop invariant - is the statement that

$$Div(a) \cap Div(b) = Div(x) \cap Div(y)$$

initial assertion $:= a \in \mathbb{N}, b \in \mathbb{N} - \{0\}, a<b$

before entering the loop we have $x = a$ & $y = b$

So clearly $Div(a) \cap Div(b) = Div(x) \cap Div(y)$

moreover, the initial assertion $\Rightarrow x = a \neq b = y$

Then, each iteration of the loop replaces x xor y by a difference of x & y so to prove p is actually a loop invariant, we must show

$$Div(x) \cap Div(y) = Div(x) \cap Div(y-x)$$

proof:

$\subseteq$: let $z \in Div(x) \cap Div(y)$, then by definition

$z | x \wedge z | y \Leftrightarrow \exists u, v \in \mathbb{Z}$ s.t.

$uz = x \wedge vz = y$ hence

$(v-u)z = vz - uz = y - x \Leftrightarrow z | y - x$

thus $z \in Div(x) \cap Div(y-x)$

$\supseteq$: let $w \in Div(x) \cap Div(y-x)$, then by definition $\exists s, t \in \mathbb{Z}$ s.t. $sw = x$ & $tw = y - x$ thus $(s+t)w = y - x + x = y$

So $w \in Div(y) \Rightarrow w \in Div(x) \cap Div(y)$ $\square$

$$\frac{(p \wedge x \neq y)\{S\}p}{\therefore\ p\{\text{while } x \neq y\ S\}(x = y \wedge p)}$$

So the above program is partially correct

Since $x = y \Rightarrow Div(x) \cap Div(y) = Div(x)$

& $\max\{Div(x)\} = x$.

Termination: consider the value $k = x + y$ every iteration of the loop reduces the value of k by at least 1 so the program terminates in fewer than k iterations.