**Program Correctness**

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

\[ \text{by definition, we cannot prove a program with infinitely many valid inputs (e.g. the Euclidean Algorithm) is correct by simply trying test inputs.} \]

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

**Proving Programs Correct**

1. \( T \Rightarrow C \) "Partial Correctness"
   - If the program terminates
   - Then the correct answer is produced

2. \( V \Rightarrow T \) "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 $p := \text{initial assertion}$ $q := \text{final assertion}$ & $S := \text{A portion of code}$ 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 $S_2$

\[ p \{ S_1 \} q \]

\[ q \{ S_2 \} r \]

\[ \therefore p \{ S \} r \]

"Composition rule" - used to break up the burden of proving $S$ is correct into two simpler tasks.

If condition then $S$

\[ (p \land \neg \text{condition}) \rightarrow q \]

\[ \therefore p \{ \text{If condition then } S \} q \]

"Conditional Statements"
If condition Then $(p \land \text{condition}) \{S_1 \} \{ p \}$  
Else $(p \land \neg \text{condition}) \{S_2 \} \{ p \}$  
"If else Statements"

"Loop Invariants"

while condition $S$  

$(p \land \text{condition}) \{S \} \{ p \}$  

:. $p \{ \text{while condition } S \} \{ \neg \text{condition} \land p \}$

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

Some Basic Examples

initial assertion

$\text{Factorial}(n > 0)$

if $n = 0$ then return 1

else return $n \cdot \text{Factorial}(n-1)$

Final assertion

We prove $p \in S \in S$

proof: We induct on $n$

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

$\neg \text{condition} \{ \neg \text{condition} \land n = 0 \} \{ \text{If } n = 0 \text{ Then return 1} \}$ output is 0!
but this is clear since $0! = 1$

**Inductive Hypothesis:**

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

**Induction Step:**

Suppose $n = k+1$ then $(k+1)! \cdot k! = (k+1)!$

by inductive hypothesis

This completes the proof of partial correctness

**Note:** The above proof by induction of the statement $n \geq 0 \implies S \{ n \}$ is basically the same as the proof using the "if-else" rule of inference

\[
\begin{align*}
\text{base case} & \quad \quad \quad (n \geq 0 \land n = 0) \implies \text{return 1} \implies n! \\
\text{inductive step} & \quad \quad \quad (n \geq 0 \land n > 0) \implies \text{return } n \cdot \text{factorial}(n-1) \implies n! \\
\therefore \quad n \geq 0 \implies S \{ n \} & \leftarrow \text{partial correctness}
\end{align*}
\]

$\therefore \; n \geq 0 \implies S$ 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 \((K+1)\text{factorial}(K)\) terminates by IH

Thus we have established "Termination"

\(\Rightarrow\) We have proven the recursive factorial function above is correct.

Exercise: Prove correctness of the following program

\[
\text{power}(a \in \mathbb{Z}; n > 0)
\]

If \( n = 0 \) then return 1

Else return \( a \cdot \text{power}(a, n-1) \)

\(\text{Output is } a^n\)

Iterative Factorial Correctness Proof

\[\begin{align*}
S_1 & \sum i := 1 \\
& \text{factorial} := 1 \\
& \text{while } i < n \\
S_2 & \sum i := i+1 \\
& \text{factorial} := \text{factorial} \cdot i \\
& \text{return factorial}
\end{align*}\]

Loop Invariant

\[p := \("\text{factorial} = i! \ & \ i \leq n"\)\]

proof: \( p \ & \ i < n \) implies

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

\[\text{factorial}_{\text{new}} = \text{factorial} \cdot (i_{\text{new}}) = i! (i+1) = (i+1)!\]

other half of \( p \)
\[
\frac{(p \land i < n) \{ S_2 \} p}{\vdash p \{ \text{while } i < n \} S_2 \{ \neg (i < n) \land p \}}
\]

Clearly, \((n \geq 1) \{ S_1 \}; S_2 \{ \text{factorial} = n! \} \)

Hence we have established partial correctness \(\square\)

For \(\square\) 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 $a, b$

- a positive integer $d$ with the following properties
  1. $d | a$ (i.e. $d \in \text{Div}(a)$)
  2. $d | b$ (i.e. $d \in \text{Div}(b)$)
  3. $\forall e \text{ satisfying (i) and (ii)}$
     
     $d \geq e$

- $d = \max\{\text{Div}(a) \cap \text{Div}(b)\}$

Claim: $a \in \mathbb{N}, b \in \mathbb{N}\setminus\{0\} \Rightarrow \gcd(a, b)$ exists & is unique

Proof:

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

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

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 $d \geq (iii)$
d satisfies (i) & (ii) ⇒ d ≤ 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)

□

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

Proof: \( \forall x, x|0 \Rightarrow b|0 \land b = \max \{ \text{Div}(b) \} \)

(\( \text{Div}(b) = \emptyset \))

(\( \text{Div}(a) \cap \text{Div}(b) = \text{Div}(a) \))

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

Proof: Lesson 15 - Idea: \( \text{Div}(a) \cap \text{Div}(b) = \text{Div}(a) \cap \text{Div}(r) \)

□

Recursive Euclidean Algorithm

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

\[
\gcd(a, b) \ a, b \in \mathbb{N} \land a < b
\]

If \( a = 0 \) Then return \( b \)
Else return \( \gcd(a, b \mod a) \)

Output satisfies (i), (ii) & (iii)

Claim: The above program is correct
"Inductive" Proof:

\[ pSq \]

If Then Else proof:

\[ \text{If } a=0 \text{ Then } S, q \]

\[ \text{Else } S2, q \]

Case: \( a=0 \)

\[ \text{Lemma 0} \]

\[ \text{Lemma 1} \]

Since \( b \equiv a < b \) one input is eventually zero reducing the problem to the above case.

**Iterative Euclidean Algorithm** (division is repeated subtraction)

\[ \begin{align*}
\text{while } x \neq y \text{ do} \\
\text{if } x < y \text{ then} \\
& x = x - y \\
\text{else} \\
& y = y - x
\end{align*} \]

**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 \( \gcd(a,b) \)

\[ P \text{- loop invariant - is the statement that} \]

\[ \text{Div}(a) \cap \text{Div}(b) = \text{Div}(x) \cap \text{Div}(y) \]

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 \( \text{Div}(a) \cap \text{Div}(b) = \text{Div}(x) \cap \text{Div}(y) \)

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

\[ \text{Div}(x) \cap \text{Div}(y) = \text{Div}(x) \cap \text{Div}(y-x) \]

**proof:**

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

\( z \mid x \land z \mid y \iff \exists u, v \in \mathbb{Z} \) s.t.

\( u \in x \land v \in y \) hence

\( (v-u)z = vz - uz = y - x \iff z \mid y - x \)

thus \( z \in \text{Div}(x) \cap \text{Div}(y-x) \)

\( \supseteq \): let \( \omega \in \text{Div}(x) \cap \text{Div}(y-x) \), then by definition \( \exists s, t \in \mathbb{Z} \) s.t.

\( s \omega = x \) & \( t \omega = y - x \) thus \((s+t)\omega = y - x + x = y \)

So \( \omega \in \text{Div}(y) \implies \omega \in \text{Div}(x) \cap \text{Div}(y) \) \( \square \)
\((p \land x \neq y) \models S \models p\)

\[
\therefore \quad p \models \text{while } x \neq y \models (x = y \land p)
\]

So the above program is partially correct since \(x = y \Rightarrow \text{Div}(x) \cap \text{Div}(y) = \text{Div}(x)\) and \(\max \{\text{Div}(x)\} = x\).

**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.