Warm-Up Problem

1. What is the definition of a Hoare triple satisfying partial correctness?
2. Recall the rule for assignment:

$$\frac{}{\{\!|\, Q[E/x] \,|\!\}\ (\mathtt{x} = E)\ \{\!|\, Q \,|\!\}}\ \text{(assignment)}$$

   Why is this the correct rule and not the following rule?

$$\frac{}{\{\!|\, Q \,|\!\}\ (\mathtt{x} = E)\ \{\!|\, Q[E/x] \,|\!\}}\ \text{(assignment)}$$

Warm-Up Problem

1. What is the definition of a Hoare triple satisfying partial correctness?
2. Recall the rule for assignment:

$$\frac{}{\{\!|\, Q[E/x] \,|\!\}\ \ (\mathtt{x} = E)\ \ \{\!|\, Q \,|\!\}} \text{ (assignment)}$$

Why is this the correct rule and not the following rule?

$$\frac{}{\{\!|\, Q \,|\!\}\ \ (\mathtt{x} = E)\ \ \{\!|\, Q[E/x] \,|\!\}} \text{ (assignment)}$$

Solution: With the second rule, triples like
$\{\!|\, x = 3 \,|\!\}$ `x=2` $\{\!|\, 2 = 3 \,|\!\}$ are provable which would allow us to prove false statements.

Program Verification Conditionals

Carmen Bruni

Lecture 18

Based on slides by Jonathan Buss, Lila Kari, Anna Lubiw and Steve Wolfman with thanks to B. Bonakdarpour, A. Gao, D. Maftuleac, C. Roberts, R. Trefler, and P. Van Beek

Last Time

- Give reasons for performing formal verification vs testing.
- Define a Hoare Triple.
- Define Partial Correctness.
- Prove that a Hoare triple is satisfied under partial correctness for a program containing assignment statements.

Learning Goals

- Understand and use implied statements as needed.
- Prove that a Hoare triple is satisfied under partial correctness for a program containing assignment and conditional statements.

Example

Show that the following Hoare triple is satisfied under partial correctness.

$(\!| \left((((2 \cdot x) + (3 \cdot y)) \geq 7) \wedge ((x + (2 \cdot y)) \geq 0)\right) |\!)$

```
x = x + y;
y = x + y;
x = x + y;
```

$(\!| \left((x \geq 5) \wedge (y \geq 0)\right) |\!)$

Example

Show that the following Hoare triple is satisfied under partial correctness.

$(\!| \ ((((2 \cdot x) + (3 \cdot y)) \geq 7) \wedge ((x + (2 \cdot y)) \geq 0)) \ |\!)$

```
x = x + y;
y = x + y;
x = x + y;
```

$(\!| \ ((x \geq 5) \wedge (y \geq 0)) \ |\!)$

Why is it that $x = 1$ and $y = 2$ satisfies the pre-condition but not the post condition and yet the above Hoare triple is satisfied under partial correctness?

Example

Show that the following Hoare triple is satisfied under partial correctness.

$(\!| \ ((((2 \cdot x) + (3 \cdot y)) \geq 7) \wedge ((x + (2 \cdot y)) \geq 0)) \ |\!)$

```
x = x + y;
y = x + y;
x = x + y;
```

$(\!| \ ((x \geq 5) \wedge (y \geq 0)) \ |\!)$

Why is it that $x = 1$ and $y = 2$ satisfies the pre-condition but not the post condition and yet the above Hoare triple is satisfied under partial correctness?

Answer: The code will actually mutate x and y! The x and y at the end is not the same as the original.

Programs with Conditional Statements

Deduction Rules for Conditionals

`if-then-else`:

$$\frac{\{\!|\, (P \wedge B) \,|\!\}\ \texttt{C}_1\ \{\!|\, Q \,|\!\} \qquad \{\!|\, (P \wedge (\neg B)) \,|\!\}\ \texttt{C}_2\ \{\!|\, Q \,|\!\}}{\{\!|\, P \,|\!\}\ \texttt{if (B) C}_1\ \texttt{else C}_2\ \{\!|\, Q \,|\!\}}\ \text{(if-then-else)}$$

`if-then` (without `else`):

$$\frac{\{\!|\, (P \wedge B) \,|\!\}\ \texttt{C}\ \{\!|\, Q \,|\!\} \qquad \big((P \wedge (\neg B)) \rightarrow Q\big)}{\{\!|\, P \,|\!\}\ \texttt{if (B) C}\ \{\!|\, Q \,|\!\}}\ \text{(if-then)}$$

Template for Conditionals With `else`

Annotated program template for `if-then-else`:

$(\!| P |\!)$

`if (` B `) {`

    $(\!| (P \wedge B) |\!)$ if-then-else

    C_1

    $(\!| Q |\!)$ *(justify depending on C_1—a "subproof")*

`} else {`

    $(\!| (P \wedge (\neg B)) |\!)$ if-then-else

    C_2

    $(\!| Q |\!)$ *(justify depending on C_2—a "subproof")*

`}`

$(\!| Q |\!)$ if-then-else [*justifies this Q, given previous two*]

Template for Conditionals Without `else`

Annotated program template for `if-then`:

$(\!| P |\!)$
`if (` B `) {`
    $(\!| (P \wedge B) |\!)$ if-then
    C
    $(\!| Q |\!)$ [*add justification based on* C]
`}`
$(\!| Q |\!)$ if-then
Implied: Proof of $\big((P \wedge (\neg B)) \rightarrow Q\big)$

Example: Conditional Code

Example: Prove the following is satisfied under partial correctness.

$\{\!|\ \mathit{true}\ |\!\}$

```
if ( max < x ) {
    max = x ;
}
```

$\{\!|\ (\mathit{max} \geq x)\ |\!\}$

$\{\!|\ P\ |\!\}$

```
if ( B ) {
    C
}
```

$\{\!|\ Q\ |\!\}$

First, let's recall our proof method....

The Steps of Creating a Proof

Three steps in doing a proof of partial correctness:

1. First annotate using the appropriate inference rules.
2. Then "back up" in the proof: add an assertion before each assignment statement, based on the assertion following the assignment.
3. Finally prove any "implieds":
   - Annotations from (1) above containing implications
   - Adjacent assertions created in step (2).

   Proofs here are written in Math 135 style. They can use Predicate Logic, basic arithmetic, or any other appropriate reasoning.

Doing the Steps

1. Add annotations for the `if-then` statement.

$(\!|\ \text{true}\ |\!)$

```
if ( max < x ) {
```

$(\!|\ (\mathit{true} \wedge (\mathit{max} < x))\ |\!)$ if-then

```
    max = x ;
```

$(\!|\ (\mathit{max} \geq x)\ |\!)$ $\longleftarrow$ *to be shown*

```
}
```

$(\!|\ (\mathit{max} \geq x)\ |\!)$ if-then

Implied: $\Big(\big(\text{true} \wedge \neg((\mathit{max} < x))\big) \rightarrow (\mathit{max} \geq x)\Big)$

Doing the Steps

1. Add annotations for the `if-then` statement.
2. "Push up" for the assignments.

$(\!|\ \text{true}\ |\!)$

```
if ( max < x ) {
```

$(\!|\ (\mathit{true} \wedge (\mathit{max} < x))\ |\!)$ if-then

$(\!|\ (x \geq x)\ |\!)$

```
    max = x ;
```

$(\!|\ (\mathit{max} \geq x)\ |\!)$ assignment

```
}
```

$(\!|\ (\mathit{max} \geq x)\ |\!)$ if-then

Implied: $\Big(\big(\text{true} \wedge \neg((\mathit{max} < x))\big) \rightarrow (\mathit{max} \geq x)\Big)$

Doing the Steps

1. Add annotations for the `if-then` statement.
2. "Push up" for the assignments.
3. Identify "implieds" to be proven.

$(\!|\ \text{true}\ |\!)$

```
if ( max < x ) {
```

$(\!|\ (\mathit{true} \wedge (\mathit{max} < x))\ |\!)$ if-then

$(\!|\ (x \geq x)\ |\!)$ Implied (a)

```
    max = x ;
```

$(\!|\ (\mathit{max} \geq x)\ |\!)$ assignment

```
}
```

$(\!|\ (\mathit{max} \geq x)\ |\!)$ if-then

Implied (b): $\Big(\big(\text{true} \wedge \neg((\mathit{max} < x))\big) \rightarrow (\mathit{max} \geq x)\Big)$

Proving "Implied" Conditions

The auxiliary "implied" proofs can be done in Math 135 style (and assuming the necessary arithmetic properties). We will write them informally, but clearly.

Proof of Implied (a):

$$\emptyset \vdash \big((\mathit{true} \wedge (\mathit{max} < x)) \to (x \geq x)\big).$$

- The statement $(x \geq x)$ is a tautology since $\geq$ is reflexive and so the required implication holds.

Implied (b)

Proof of Implied (b): Show $\emptyset \vdash \big((P \wedge (\neg B)) \rightarrow Q\big)$, which in this case is

$$\emptyset \vdash \Big(\big(\mathit{true} \wedge (\neg(\mathit{max} < x))\big) \rightarrow (\mathit{max} \geq x)\Big).$$

- The hypothesis, $\big(\mathit{true} \wedge (\neg(\mathit{max} < x))\big)$ can be simplified to $(\neg(\mathit{max} < x))$.
- Then by properties of $\neg$, $<$ and $\geq$, the conclusion, $(\mathit{max} \geq x)$, follows.

Example 2 for Conditionals

Prove the following is satisfied under partial correctness.

$(\!|\,\mathsf{true}\,|\!)$

```
if ( x > y ) {
    max = x;
} else {
    max = y;
}
```

$(\!|\,(((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\,|\!)$

Example 2: Annotated Code

$\{\!|\ \mathit{true}\ |\!\}$
`if ( x > y ) {`
    $\{\!|\ (x > y)\ |\!\}$ if-then-else

    `max = x ;`
    $\{\!|\ (((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\ |\!\}$
`} else {`
    $\{\!|\ (\neg(x > y))\ |\!\}$ if-then-else

    `max = y ;`
    $\{\!|\ (((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\ |\!\}$
`}`
$\{\!|\ (((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\ |\!\}$ if-then-else

Example 2: Annotated Code

$\{\!|\ \mathit{true}\ |\!\}$

```
if ( x > y ) {
```

$\{\!|\ (x > y)\ |\!\}$ — if-then-else

$\{\!|\ (((x > y) \wedge (x = x)) \vee ((x \leq y) \wedge (x = y)))\ |\!\}$

```
    max = x ;
```

$\{\!|\ (((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\ |\!\}$ — assignment

```
} else {
```

$\{\!|\ (\neg(x > y))\ |\!\}$ — if-then-else

$\{\!|\ (((x > y) \wedge (y = x)) \vee ((x \leq y) \wedge (y = y)))\ |\!\}$

```
    max = y ;
```

$\{\!|\ (((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\ |\!\}$ — assignment

```
}
```

$\{\!|\ (((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\ |\!\}$ — if-then-else

Example 2: Annotated Code

$\{\!|\ \mathit{true}\ |\!\}$

```
if ( x > y ) {
```

$\{\!\|\ (x > y)\ \|\!\}$	if-then-else
$\{\!\|\ (((x > y) \wedge (x = x)) \vee ((x \leq y) \wedge (x = y)))\ \|\!\}$	implied (a)

```
    max = x ;
```

$\{\!\|\ (((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\ \|\!\}$	assignment

```
} else {
```

$\{\!\|\ (\neg(x > y))\ \|\!\}$	if-then-else
$\{\!\|\ (((x > y) \wedge (y = x)) \vee ((x \leq y) \wedge (y = y)))\ \|\!\}$	implied (b)

```
    max = y ;
```

$\{\!\|\ (((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\ \|\!\}$	assignment

```
}
```

$\{\!\|\ (((x > y) \wedge (\mathit{max} = x)) \vee ((x \leq y) \wedge (\mathit{max} = y)))\ \|\!\}$	if-then-else

Example 2: Implied Conditions

(a) Prove $\Big((x > y) \to \big(((x > y) \wedge (x = x)) \vee ((x \leq y) \wedge (x = y))\big)\Big)$.

- From $(x > y)$ and from the trivially true $(x = x)$, it follows that $\big((x > y) \wedge (x = x)\big)$.
- From this it follows that $\big(((x > y) \wedge (x = x)) \vee ((x \leq y) \wedge (x = y))\big)$.

(b) Prove $\Big((x \leq y) \to \big(((x > y) \wedge (y = x)) \vee ((x \leq y) \wedge (y = y))\big)\Big)$.

- From $(x \leq y)$ and from the trivially true $(y = y)$, it follows that $\big((x > y) \wedge (y = x)\big)$.
- From this it follows that $\big(((x > y) \wedge (y = x)) \vee ((x \leq y) \wedge (y = y))\big)$.

`While`-Loops and Total Correctness

Total correctness

A triple $(\!| P |\!)\ C\ (\!| Q |\!)$ is satisfied under total correctness, denoted

$$\models_{\mathtt{tot}} (\!| P |\!)\ C\ (\!| Q |\!)\ ,$$

if and only if

> for every state s that satisfies P,
>
> execution of C starting from state s terminates,
>
> and the resulting state s' satisfies Q.

$$\text{Total Correctness} = \text{Partial Correctness} + \text{Termination}$$

Examples for Partial and Total Correctness

Example 1. Total correctness satisfied:

$$(\!| (x = 1) |\!)$$
$$\texttt{y = x ;}$$
$$(\!| (y = 1) |\!)$$

Example 2. Neither total nor partial correctness:

$$(\!| (x = 1) |\!)$$
$$\texttt{y = x ;}$$
$$(\!| (y = 2) |\!)$$

Examples for Partial and Total Correctness

Example 3. Infinite loop (partial correctness)

$(\!| (x = 1) |\!)$
```
while (true) {
    x = 0 ;
}
```
$(\!| (x > 0) |\!)$

Example 4. Total correctness

$(\!| (x \geq 0) |\!)$
```
y = 1 ;
z = 0 ;
while (z != x) {
    z = z + 1 ;
    y = y * z ;
}
```
$(\!| (y = x!) |\!)$

What happens if we remove the precondition?

Examples for Partial and Total Correctness

Example 5. No correctness, because input altered ("consumed")

$(\!|\, \text{true} \,|\!)$

```
y = 1 ;
while (x != 0) {
    y = y * x ;
    x = x - 1 ;
}
```

$(\!|\, (y = x!) \,|\!)$

Proving Correctness: Recap and Overview

- Total correctness is our goal.
- We usually prove it by proving partial correctness and termination separately.
  - For partial correctness, we introduced sound inference rules.
  - For total correctness, we shall use *ad hoc* reasoning, which suffices for our examples.
    (In general, total correctness is undecidable.)

Our focus on partial correctness may seem strange. It's not the condition we want to justify.

But experience has shown it is useful to think about partial correctness separately from termination.

Inference Rule: Partial-`while`

"Partial while": do not (yet) require termination.

$$\frac{\{\!|\, I \wedge B \,|\!\}\ C\ \{\!|\, I \,|\!\}}{\{\!|\, I \,|\!\}\ \texttt{while}\ (B)\ C\ \{\!|\, I \wedge \neg B \,|\!\}}\ \text{(partial-}\textit{while}\text{)}$$

In words:

> If the code C satisfies the triple $\{\!|\, I \wedge B \,|\!\}\ C\ \{\!|\, I \,|\!\}$,
> and I is true at the start of the `while`-loop,
> then no matter how many times we execute C,
> condition I will still be true.

Condition I is called a *loop invariant*.

After the `while`-loop terminates, $\neg B$ is also true.

Annotations for Partial-`while`

$(\!| P |\!)$

$(\!| I |\!)$ Implied (a)

`while ( B ) {`

  $(\!| I \wedge B |\!)$ partial-`while`

  C

  $(\!| I |\!)$ $\longleftarrow$ *to be justified, based on* C

  }

$(\!| I \wedge \neg B |\!)$ partial-`while`

$(\!| Q |\!)$ Implied (b)

(a) Prove $P \rightarrow I$ (precondition P implies the loop invariant)

(b) Prove $(I \wedge \neg B) \rightarrow Q$ (exit condition implies postcondition)

We need to determine I!!

Loop Invariants

A *loop invariant* is an assertion (condition) that is true both *before* and *after* each execution of the body of a loop.

- True before the `while`-loop begins.
- True after the `while`-loop ends.
- Expresses a relationship among the variables used within the body of the loop. Some of these variables will have their values changed within the loop.
- An invariant may or may not be useful in proving termination (to discuss later).

Example: Finding a loop invariant

$(\!| \, x \geq 0 \, |\!)$

```
y = 1 ;
z = 0 ;
while (z != x) {
    z = z + 1 ;
    y = y * z ;
}
```

$(\!| \, y = x! \, |\!)$

Example: Finding a loop invariant

$(\!| \, x \geq 0 \, |\!)$

```
y = 1 ;
z = 0 ;
while (z != x) {
    z = z + 1 ;
    y = y * z ;
}
```

$(\!| \, y = x! \, |\!)$

At the `while` statement:

x	y	z	$z \neq x$
5	1	0	true
5	1	1	true
5	2	2	true
5	6	3	true
5	24	4	true
5	120	5	false

Example: Finding a loop invariant

$(\!| x \geq 0 |\!)$

```
y = 1 ;
z = 0 ;
→ while (z != x) {
      z = z + 1 ;
      y = y * z ;
  }
```

$(\!| y = x! |\!)$

At the `while` statement:

x	y	z	$z \neq x$
5	1	0	true
5	1	1	true
5	2	2	true
5	6	3	true
5	24	4	true
5	120	5	false

From the trace and the post-condition,
a candidate loop invariant is $y = z!$

Why are $y \geq z$ or $x \geq 0$ not useful?

These do not involve the loop-termination condition.

Annotations Inside a `while`-Loop

1. First annotate code using the while-loop inference rule, and any other control rules, such as `if-then`.
2. Then work bottom-up ("push up") through program code.
   - Apply inference rule appropriate for the specific line of code, or
   - Note a new assertion ("implied") to be proven separately.
3. Prove the implied assertions using the inference rules of ordinary logic.

Example: annotations for partial-`while`

Annotate by partial-`while`, with chosen invariant $(y = z!)$.

$(\!| x \geq 0 |\!)$

```
y = 1 ;

z = 0 ;
```
$(\!| y = z! |\!)$ *[justification required]*

```
while (z != x) {
```
$(\!| (y = z!) \wedge \neg(z = x) |\!)$ partial-`while` $((\!| I \wedge B |\!))$

```
    z = z + 1 ;

    y = y * z ;
```
$(\!| y = z! |\!)$ *[justification required]*

```
}
```
$(\!| y = z! \wedge (z = x) |\!)$ partial-`while` $((\!| I \wedge \neg B |\!))$

$(\!| y = x! |\!)$

Example: annotations for partial-`while`

Annotate assignment statements (bottom-up).

$(\!| x \geq 0 |\!)$
$(\!| 1 = 0! |\!)$
`y = 1 ;`
$(\!| y = 0! |\!)$ assignment
`z = 0 ;`
$(\!| y = z! |\!)$ assignment
`while (z != x) {`
  $(\!| (y = z!) \wedge \neg(z = x) |\!)$ partial-`while`
  $(\!| y(z+1) = (z+1)! |\!)$
  `z = z + 1 ;`
  $(\!| yz = z! |\!)$ assignment
  `y = y * z ;`
  $(\!| y = z! |\!)$ assignment
`}`
$(\!| y = z! \wedge (z = x) |\!)$ partial-`while`
$(\!| y = x! |\!)$

Example: annotations for partial-`while`

Note the required implied conditions.

$\{\!\mid x \geq 0 \mid\!\}$	
$\{\!\mid 1 = 0! \mid\!\}$	implied (a)
`y = 1 ;`	
$\{\!\mid y = 0! \mid\!\}$	assignment
`z = 0 ;`	
$\{\!\mid y = z! \mid\!\}$	assignment
`while (z != x) {`	
$\quad\{\!\mid (y = z!) \wedge \neg(z = x) \mid\!\}$	partial-`while`
$\quad\{\!\mid y(z+1) = (z+1)! \mid\!\}$	implied (b)
`  z = z + 1 ;`	
$\quad\{\!\mid yz = z! \mid\!\}$	assignment
`  y = y * z ;`	
$\quad\{\!\mid y = z! \mid\!\}$	assignment
`}`	
$\{\!\mid y = z! \wedge (z = x) \mid\!\}$	partial-`while`
$\{\!\mid y = x! \mid\!\}$	implied (c)

Example: implied conditions (a) and (c)

Proof of implied (a): $(x \geq 0) \vdash (1 = 0!)$.

By definition of factorial.

Proof of implied (c): $\Big((y = z!) \wedge (z = x)\Big) \vdash (y = x!)$.

1.	$((y = z!) \wedge (z = x))$	Premise
2.	$(y = z!)$	$\wedge$e : 1
3.	$(z = x)$	$\wedge$e : 1
4.	$(z! = x!)$	EQSubs $[f(u) = u!]$: 3
5.	$(y = x!)$	EQTrans : 2, 4

Example: implied condition (b)

Proof of implied (b):

$$\Big((y = z!) \land \neg(z = x)\Big) \vdash (z+1)\, y = (z+1)! \ .$$

1.	$y = z! \land z \neq x$	premise
2.	$y = z!$	$\land$e: 1
3.	$(z+1)\, y = (z+1)\, z!$	EqSubs: 2
4.	$(z+1)\, z! = (z+1)!$	def. of factorial: 3
5.	$(z+1)\, y = (z+1)!$	EqTrans: 3, 4

Example 2 (Partial-`while`)

Prove the following is satisfied under partial correctness.

$(\!| n \geq 0 \wedge a \geq 0 |\!)$

```
s = 1 ;
i = 0 ;
while (i < n) {
    s = s * a ;
    i = i + 1 ;
}
```

$(\!| s = a^n |\!)$

Example 2 (Partial-`while`)

Prove the following is satisfied under partial correctness.

$(\!| n \geq 0 \wedge a \geq 0 |\!)$

```
s = 1 ;
i = 0 ;
while (i < n) {
    s = s * a ;
    i = i + 1 ;
}
```

$(\!| s = a^n |\!)$

Trace of the loop:

a	n	i	s
2	3	0	1
2	3	1	1∗2
2	3	2	1∗2∗2
2	3	3	1∗2∗2∗2

Example 2 (Partial-`while`)

Prove the following is satisfied under partial correctness.

$(\!| n \geq 0 \wedge a \geq 0 |\!)$

```
s = 1 ;
i = 0 ;
while (i < n) {
    s = s * a ;
    i = i + 1 ;
}
```

$(\!| s = a^n |\!)$

Trace of the loop:

a	n	i	s
2	3	0	1
2	3	1	1∗2
2	3	2	1∗2∗2
2	3	3	1∗2∗2∗2

Candidate for the loop invariant: $s = a^i$.

Example 2: Testing the invariant

Using $s = a^i$ as an invariant yields the annotations shown at right.

Next, we want to

- Push up for assignments
- Prove the implications

But: implied (c) is false!

We must use a different invariant.

$\{\!| n \geq 0 \wedge a \geq 0 |\!\}$
$\{\!|$ ... $|\!\}$
`s = 1 ;`
$\{\!|$ ... $|\!\}$
`i = 0 ;`
$\{\!| s = a^i |\!\}$
`while (i < n) {`
  $\{\!| s = a^i \wedge i < n |\!\}$ partial-`while`
  $\{\!|$ ... $|\!\}$
  `s = s * a ;`
  $\{\!|$ ... $|\!\}$
  `i = i + 1 ;`
  $\{\!| s = a^i |\!\}$
`}`
$\{\!| s = a^i \wedge i \geq n |\!\}$ partial-`while`
$\{\!| s = a^n |\!\}$ implied (c)

Example 2: Adjusted invariant

Try a new invariant:

$$s = a^i \wedge i \leq n \ .$$

Now the "implied" conditions are actually true, and the proof can succeed.

$\{\!\mid n \geq 0 \wedge a \geq 0 \mid\!\}$	
$\{\!\mid 1 = a^0 \wedge 0 \leq n \mid\!\}$	implied (a)
`s = 1 ;`	
$\{\!\mid s = a^0 \wedge 0 \leq n \mid\!\}$	assignment
`i = 0 ;`	
$\{\!\mid s = a^i \wedge i \leq n \mid\!\}$	assignment
`while (i < n) {`	
$\quad \{\!\mid s = a^i \wedge i \leq n \wedge i < n \mid\!\}$	partial-`while`
$\quad \{\!\mid s \cdot a = a^{i+1} \wedge i + 1 \leq n \mid\!\}$	implied (b)
`  s = s * a ;`	
$\quad \{\!\mid s = a^{i+1} \wedge i + 1 \leq n \mid\!\}$	assignment
`  i = i + 1 ;`	
$\quad \{\!\mid s = a^i \wedge i \leq n \mid\!\}$	assignment
`}`	
$\{\!\mid s = a^i \wedge i \leq n \wedge i \geq n \mid\!\}$	partial-`while`
$\{\!\mid s = a^n \mid\!\}$	implied (c)

Total Correctness (Termination)

Total Correctness = Partial Correctness + Termination

Only `while`-loops can be responsible for non-termination in our programming language.

(In general, recursion can also cause it).

Proving termination:
For each `while`-loop in the program,

> Identify an integer expression which is *always* non-negative and whose value *decreases* every time through the while-loop.

Example For Total Correctness

The code below has a "*loop guard*" of $z \neq x$,
which is equivalent to $x - z \neq 0$.

What happens to the value of $x - z$ during execution?

$(\!| x \geq 0 |\!)$

```
y = 1 ;
z = 0 ;
```

At start of loop: $x - z = x \geq 0$

```
while ( z != x ) {
    z = z + 1 ;
    y = y * z ;
}
```

$(\!| y = x! |\!)$

Example For Total Correctness

The code below has a "*loop guard*" of $z \neq x$,
which is equivalent to $x - z \neq 0$.

What happens to the value of $x - z$ during execution?

$(\!| x \geq 0 |\!)$

```
y = 1 ;
z = 0 ;
```

At start of loop: $x - z = x \geq 0$

```
while ( z != x ) {
    z = z + 1 ;
    y = y * z ;
}
```

$x - z$ decreases by 1

$(\!| y = x! |\!)$

Example For Total Correctness

The code below has a "*loop guard*" of $z \neq x$,
which is equivalent to $x - z \neq 0$.

What happens to the value of $x - z$ during execution?

$(\!| x \geq 0 |\!)$
```
y = 1 ;
z = 0 ;
```
At start of loop: $x - z = x \geq 0$
```
while ( z != x ) {
    z = z + 1 ;
    y = y * z ;
}
```
$x - z$ decreases by 1

$x - z$ unchanged

$(\!| y = x! |\!)$

Example For Total Correctness

The code below has a "*loop guard*" of $z \neq x$,
which is equivalent to $x - z \neq 0$.

What happens to the value of $x - z$ during execution?

$\{\!|\, x \geq 0 \,|\!\}$
```
y = 1 ;
z = 0 ;
```
At start of loop: $x - z = x \geq 0$
```
while ( z != x ) {
    z = z + 1 ;
    y = y * z ;
}
```
$x - z$ decreases by 1

$x - z$ unchanged

$\{\!|\, y = x! \,|\!\}$

Thus the value of $x - z$ will eventually reach 0.
The loop then exits and the program terminates.

Proof of Total Correctness

We chose an expression $x - z$ (called the *variant*).

At the start of the loop, $x - z \geq 0$:

- Precondition: $x \geq 0$.
- Assignment $z \leftarrow 0$.

Each time through the loop:

- x doesn't change: no assignment to it.
- z increases by 1, by assignment.
- Thus $x - z$ decreases by 1.

Thus the value of $x - z$ will eventually reach 0.

When $x - z = 0$, the guard $z \mathrel{!}= x$ ends the loop.