Hoare Logic: Total Correctness

COMP1600 / COMP6260

Victor Rivera Dirk Pattinson
Australian National University

Semester 2, 2021

Recall: Hoare Logic

Basic Ingredient. Hoare-triples

$$\{P\}\ S\ \{Q\}$$

- P and Q are predicates (formulae)
- S is a code (fragment)

Example. $\{x \geq 0\}$ `while (x>0) do x := x-1` $\{x = 0\}$

Meaning. *If* we run S from a state that satisfies precondition P *and* if S terminates, then the post-state will satisfy Q.

Hoare Logic

Idea. *Proof Rules* that allow us to prove all *true* triples.

Assignment

$$\frac{}{\{Q(e)\}\ \texttt{x := e}\ \{Q(x)\}}$$

Precondition Strengthening / Postcondition Weakening

$$\frac{P_s \to P_w \qquad \{P_w\}\ \texttt{S}\ \{Q\}}{\{P_s\}\ \texttt{S}\ \{Q\}} \qquad \frac{\{P\}\ \texttt{S}\ \{Q_s\} \qquad Q_s \to Q_w}{\{P\}\ \texttt{S}\ \{Q_w\}}$$

Sequence.

$$\frac{\{P\}\ \texttt{S}_1\ \{Q\} \qquad \{Q\}\ \texttt{S}_2\ \{R\}}{\{P\}\ \texttt{S}_1;\texttt{S}_2\ \{R\}}$$

Conditional.

$$\frac{\{P \wedge b\}\ \texttt{S}_1\ \{Q\} \qquad \{P \wedge \neg b\}\ \texttt{S}_2\ \{Q\}}{\{P\}\ \texttt{if b then S}_1\ \texttt{else S}_2\ \{Q\}}$$

2 / 49

Proof Rule for While Loops (Rule 6/6)

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}}$$

- I is called the **loop invariant**.
- I is true before we encounter the while statement, and remains true each time around the loop (although not necessarily midway *during* execution of the loop body).
- If the loop terminates the control condition must be false, so $\neg b$ appears in the postcondition.
- For the body of the loop S to execute, b needs to be true, so it appears in the precondition.

3 / 49

Soundness of the While Rule w.r.t. the semantics

$$\frac{\{I \wedge b\}\ \ S\ \ \{I\}}{\{I\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{I \wedge \neg b\}}$$

Soundness. If the premise is true, then so is the conclusion.

- assume that I holds in the initial state.
- if b is false, nothing happens, so $I \wedge \neg b$ is true in post-state.
- if b is true, then (by premise) I holds at end of each iteration
- *assuming* that the loop terminates, b becomes false (and I still holds)

Q. What about non-termination?

Applying the While Rule

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

Difficult bit. Finding the right invariant.

- This requires *intuition* and *practice*. There is no automated way of doing this.

Easy bit. Establishing the desired postcondition

- The postcondition we get after applying our rule has form $I \wedge \neg b$. But if $I \wedge \neg b \to Q$, we can use *postcondition weakening*.

Easy bit. Establishing the desired precondition

- The precondition we get after applying our rule has form I. But if $P \to I$, we can use *precondition strengtening*.

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

$\{P\}$ **while** b **do** S $\{Q\}$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

$\{I\}$ **while** b **do** S $\{I \wedge \neg b\}$

$\{P\}$ **while** b **do** S $\{Q\}$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\}\ \ S\ \ \{I\}}{\{I\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{I \wedge \neg b\}} \qquad \{P\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{Q\}$$

$\{I \wedge b\}\ \ S\ \ \{I\}$
$\{I\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{I \wedge \neg b\}$

$\{P\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{Q\}$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\}\ \ S\ \ \{I\}}{\{I\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{I \wedge \neg b\}} \qquad \{P\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{Q\}$$

1. $\{I \wedge b\}\ \ S\ \ \{I\}$
   $\{I\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{I \wedge \neg b\}$

$\{P\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{Q\}$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

1. $\{I \wedge b\} \;\; S \;\; \{I\}$
2. $\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}$ (While Loop, 1)

$\{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

1. $\{I \wedge b\} \;\; S \;\; \{I\}$
2. $\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}$ (While Loop, 1)

   $I \wedge \neg b \rightarrow Q$

$\{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$

Applying the While Rule (schematic proof)

$$\frac{\{I \land b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \land \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

1. $\{I \land b\} \;\; S \;\; \{I\}$
2. $\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \land \neg b\}$ (While Loop, 1)
3. $I \land \neg b \to Q$ (Logic)

$\{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

1. $\{I \wedge b\} \;\; S \;\; \{I\}$
2. $\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}$ (While Loop, 1)
3. $I \wedge \neg b \rightarrow Q$ (Logic)
4. $\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$ (Postcondition Weakening, 2, 3)

$$\{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

1. $\{I \wedge b\} \;\; S \;\; \{I\}$
2. $\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}$ (While Loop, 1)
3. $I \wedge \neg b \rightarrow Q$ (Logic)
4. $\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$ (Postcondition Weakening, 2, 3)

$P \rightarrow I$

$\{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\}\ \ S\ \ \{I\}}{\{I\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{I \wedge \neg b\}} \qquad \{P\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{Q\}$$

1. $\{I \wedge b\}\ \ S\ \ \{I\}$
2. $\{I\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{I \wedge \neg b\}$ (While Loop, 1)
3. $I \wedge \neg b \to Q$ (Logic)
4. $\{I\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{Q\}$ (Postcondition Weakening, 2, 3)
5. $P \to I$ (Logic)

$\{P\}\ \ \textbf{while}\ \ b\ \ \textbf{do}\ \ S\ \ \{Q\}$

Applying the While Rule (schematic proof)

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}} \qquad \{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$$

1. $\{I \wedge b\} \;\; S \;\; \{I\}$
2. $\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}$ (While Loop, 1)
3. $I \wedge \neg b \rightarrow Q$ (Logic)
4. $\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$ (Postcondition Weakening, 2, 3)
5. $P \rightarrow I$ (Logic)
6. $\{P\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{Q\}$ (Precondition Strengthening, 4, 5)

Example

Goal. Find condition I to prove that:

$$\{n > 3\}\ \texttt{while n>0 do n := n-1}\ \{n = 0\}$$

Observation. Need to prove the above using while-rule, i.e.

$$\frac{\{I \wedge b\}\ \texttt{n} := \texttt{n} - 1\ \{I\}}{\{I\}\ \texttt{while (n>0) do n:= n-1}\ \{I \wedge n \leq 0\}}$$

Want. Loop invariant I such that

- It is implied by the precondition:

$$n > 3 \to I$$

- if the loop terminates (i.e. $n > 0$ is false), then I should imply the postcondition:

$$I \wedge n \leq 0 \to n = 0$$

- If I is true and the body is executed, I is true afterwards:

$$\{I \wedge n > 0\}\ \texttt{n} := \texttt{n} - 1\ \{I\}$$

Example (cont.)

Goal: Find condition I to prove that:

$$\{n > 3\} \texttt{ while n>0 do n := n-1 } \{n = 0\}$$

Loop Invariant: $I \equiv (n \geq 0)$, we have

- $n > 3 \rightarrow n \geq 0$,
- $n \geq 0 \wedge n \leq 0 \rightarrow n = 0$, and
- $\{n \geq 0 \wedge n > 0\}$ n := n − 1 $\{n \geq 0\}$.

Example, Formally

1. $\{n - 1 \geq 0\}$ $\texttt{n := n - 1}$ $\{n \geq 0\}$ (Assignment)
2. $n \geq 0 \wedge n > 0 \rightarrow n - 1 \geq 0$ (Logic)
3. $\{n \geq 0 \wedge n > 0\}$ $\texttt{n := n - 1}$ $\{n \geq 0\}$ (Prec. Strength., 1, 2)
4. $\{n \geq 0\}$ $\texttt{while (n > 0) do n := n - 1}$ $\{n \geq 0 \wedge \neg(n > 0)\}$ (While Loop, 3)
5. $n > 3 \rightarrow n \geq 0$ (Logic)
6. $\{n > 3\}$ $\texttt{while (n > 0) do n := n - 1}$ $\{n \geq 0 \wedge \neg(n > 0)\}$ (Prec. Strength., 4, 5)
7. $n = 0 \leftrightarrow n \geq 0 \wedge \neg(n > 0)$ (Logic)
8. $\{n \geq 0\}$ $\texttt{while (n > 0) do n := n - 1}$ $\{n = 0\}$ (Post. Equiv., 6, 7)

Other Invariants

- e.g. *true* or $n = 0$
- both are invariants, and give $n = 0$ as postcondition
- but $n \geq 0$ is *better* (weaker) as it is more general.

Let's Prove a Program!

Program (with specification):


```
{True}
i:=0;
s:=0;
while (i ≠ n) do
  i:=i+1;
  s:=s+(2*i-1)
{s = n²}
```


(The sum of the first n odd numbers is n^2)

Goal: prove

$$\{True\}\ Program\ \{s = n^2\}$$

A Very Informal Analysis

Let's look at some examples:

$$1 = 1 = 1^2$$

$$1 + 3 = 4 = 2^2$$

$$1 + 3 + 5 = 9 = 3^2$$

$$1 + 3 + 5 + 7 = 16 = 4^2 \ldots$$

It looks OK - let's see if we can prove it!

Goal: prove

$$\{\mathit{True}\}\ \mathit{Program}\ \{s = n^2\}$$

How can we prove it?

First Task. Find a loop invariant I. (NB: S and s are different!)

Post condition and loop condition:

$$\frac{\{I \wedge b\} \;\; S \;\; \{I\}}{\{I\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \{I \wedge \neg b\}}$$


```
while (i ≠ n) do
   i:=i+1;           (1, 2, 3, ...)
   s:=s+(2*i-1)      (1, 4, 9, ...)
{s = n²}
```


Want. $(I \wedge i = n) \rightarrow (s = n^2)$ to apply postcondweak

Loop Body. Each time `i` increments, `s` moves to next square number.

Invariant. $I \equiv s = i^2$.

Check I as $(s = i^2)$ is an invariant: prove $\{I\}S\{I\}$

$$\frac{\{s = i^2\}\ i := i + 1\ \{Q\} \quad \{Q\}\ s := s + (2 * i - 1)\ \{s = i^2\}}{\{s = i^2\}\ i := i + 1; s := s + (2 * i - 1)\ \{s = i^2\}}\ \text{Seq}$$

Using the *assignment axiom* and the *sequence rule*:

1. $\{Q\}$ `s:=s+(2*i-1)` $\{s = i^2\}$
2.
3. $\{s = i^2\}$ `i:=i+1` $\{Q\}$
4. $\{s = i^2\}$ `i:=i+1; s:=s+(2*i-1)` $\{s = i^2\}$ (Sequence, 3, 1)

Check I as $(s = i^2)$ is an invariant: prove $\{I\}S\{I\}$

$$\frac{\{s = i^2\}\ \ i := i+1\ \ \{Q\} \qquad \{Q\}\ \ s := s + (2 * i - 1)\ \ \{s = i^2\}}{\{s = i^2\}\ \ i := i+1;\ s := s + (2 * i - 1)\ \ \{s = i^2\}}\ \text{Seq}$$

Q is $\{s + (2 * i - 1) = i^2\}$

Using the *assignment axiom* and the *sequence rule*:

1. $\{s + (2 * i - 1) = i^2\}$ `s:=s+(2*i-1)` $\{s = i^2\}$ (Assignment)
2.
3. $\{s = i^2\}$ `i:=i+1` $\{s + (2 * i - 1) = i^2\}$
4. $\{s = i^2\}$ `i:=i+1; s:=s+(2*i-1)` $\{s = i^2\}$ (Sequence, 3, 1)

Check I as $(s = i^2)$ is an invariant: prove $\{I\}S\{I\}$

$$\frac{\{s = i^2\}\ \ i := i + 1\ \ \{Q\} \qquad \{Q\}\ \ s := s + (2 * i - 1)\ \ \{s = i^2\}}{\{s = i^2\}\ \ i := i + 1; s := s + (2 * i - 1)\ \ \{s = i^2\}}\ \text{Seq}$$

Q is $\{s + (2 * i - 1) = i^2\}$

Using the *assignment axiom* and the *sequence rule*:

1. $\{s + (2 * i - 1) = i^2\}$ `s:=s+(2*i-1)` $\{s = i^2\}$ (Assignment)
2. $\{s + (2 * (i + 1) - 1) = (i + 1)^2\}$ `i:=i+1` $\{s + (2 * i - 1) = i^2\}$ (Assignment)
3. $\{s = i^2\}$ `i:=i+1` $\{s + (2 * i - 1) = i^2\}$ (Prec. Equiv., 2)
4. $\{s = i^2\}$ `i:=i+1; s:=s+(2*i-1)` $\{s = i^2\}$ (Sequence, 3, 1)

So far, so good. (I as $(s = i^2)$ **is an invariant**.)

Completing the Proof of $\{True\}$ *Program* $\{s = n^2\}$

6 Strengthen the precondition to match the While rule premise $\{I \wedge b\}\ S\ \{I\}$

$$\{(s = i^2) \wedge (i \neq n)\}\ \texttt{i:=i+1; s:=s+(2*i-1)}\ \{s = i^2\}$$

7 Apply the While rule and postcondition equiv: $s = i^2 \wedge i = n \leftrightarrow s = n^2$

$$\{s = i^2\}\ \texttt{while ... s:=s+(2*i-1)}\ \{s = n^2\}$$

8 Check that the **initialisation establishes the invariant:**

$$\frac{\{0 = 0^2\}\ \texttt{i:=0}\ \{0 = i^2\} \qquad \{0 = i^2\}\ \texttt{s:=0}\ \{s = i^2\}}{\{0 = 0^2\}\ \texttt{i:=0; s:=0}\ \{s = i^2\}}$$

9 $(0 = 0^2)\ \leftrightarrow\ True$, so putting it all together with Sequencing we have

$$\{True\}\ \texttt{i} := 0\ ;\ \texttt{s} := 0\ ;\ \texttt{while}\ (\texttt{i} \neq \texttt{n})\ \texttt{do S}\ \{s = n^2\}$$

What about Termination?

Hoare Logic (in this form) proves *partial correctness*.

Example: `while 1+1 = 2 do x:=0`

- This will loop forever!
- can still prove things about it

Exercise:

$$\{True\}\ \texttt{while 1+1 = 2 do x:=0}\ \{False\}$$

Termination:

- remember functional programs? Something must decrease
- need *loop variant* (later)

Are the Rules Complete?

So far. Have a *very simple* language

- new rules for arrays, **for**-loops, exceptions, . . .

Focus here. Soundness

- every provable Hoare triple is (semantically) true
- soundness holds but terms and conditions apply
- with these assumptions, also have *completeness*, i.e. every true Hoare triple is provable.

Completeness. if $\{P\}\ S\ \{Q\}$ is true then $\{P\}\ S\ \{Q\}$ is provable

What are these Assumptions?

- The language we use for expressions in our programs is the same as the language we use in our pre- and postconditions (in our case, basic arithmetic).
- We assumed no **aliasing** of variables. (In most real languages we can have multiple names for the one piece of memory.)

How is *aliasing* a problem?

What are these Assumptions?

- The language we use for expressions in our programs is the same as the language we use in our pre- and postconditions (in our case: basic arithmetic).
- We assumed no **aliasing** of variables. (In most real languages we can have multiple names for the one piece of memory.)

How is *aliasing* a problem?

- Suppose x and y refer to the same cell of memory.
  - We get $\{y + 1 = 5 \ \wedge \ y = 5\}$ `x:=y+1` $\{x = 5 \ \wedge \ y = 5\}$ (Assignment)
  - i.e. $\{y = 4 \ \wedge \ y = 5\}$ `x:=y+1` $\{x = 5 \ \wedge \ y = 5\}$
  - i.e. if initial state satisfies *False* and `x:=y+1` terminates then final state satisfies $\{x = 5 \ \wedge \ y = 5\}$ (but also works for $x = 6 \wedge y = 6$)

which makes a mockery of our calculus since it proves rubbish!

Finding a Proof

Example


```
{ n >= 0 }
  f := 1;
  i := n;
  while (i > 0) do
    f := f * i;
    i := i-1;
{ f = n! }
```

Finding a Proof

Annotating the program: $I = f * i! = n! \wedge i \geq 0$

```
{ n >= 0 }
  f := 1;
{ f = 1 /\ n >= 0 }             -- provable with assignment
  i := n;
{ f = 1 /\ i = n /\ n >= 0}  -- provable with assignment
==>                             -- use postcond weakening
{ I }                           -- general premise of while rule
  while (i > 0) do
    { I /\ i > 0 }              -- proof obligation for loop body
      f := f * i;               -- n,  n * (n-1), n * (n-1) * (n-2) .
      i := i-1;                 -- n-1, n-2,      n-3,              .
    { I }                       -- up until here.
{ I /\ not(i > 0) }             -- general conclusion of while rule
==>                             -- use postcond weakening
{ f = n! }
```

From Annotated Programs to Proofs

Initialisation Part

```
{ n >= 0 }
  f := 1;
{ f = 1 /\ n >= 0 }     -- provable with assignment
  i := n;
{ f = 1 /\ i = n /\ n >= 0 }  -- provable with assignment
```


1. $\{f = 1 \wedge n = n \wedge n \geq 0\}$ `i := n` $\{f = 1 \wedge i = n \wedge n \geq 0\}$ (Assignment)
2. $\{1 = 1 \wedge n = n \wedge n \geq 0\}$ `f := 1` $\{f = 1 \wedge n = n \wedge n \geq 0\}$ (Assignment)
3. $n \geq 0 \leftrightarrow 1 = 1 \wedge n = n \wedge n \geq 0$ (Logic)
4. $\{n \geq 0\}$ `f := 1` $\{f = 1 \wedge n = n \wedge n \geq 0\}$ (Prec. Equiv. 2, 3)
5. $\{n \geq 0\}$ `f := 1; i := n` $\{f = 1 \wedge i = n \wedge n \geq 0\}$ (Sequence, 1, 4)

From Invariant to Loop Body

```
{ I }                    -- general premis of while rule
  while (i > 0) do
    { I /\ i > 0 }       -- proof obligation for loop body
      f := f * i;        -- n,   n * (n-1), n * (n-1) * (n-2) ...
      i := i-1;          -- n-1, n-2,       n-3,              ...
    { I }                -- up until here.
{ I /\ not(i > 0) }      -- general conclusion of while rule
```


Invariant. $I = f * i! = n! \wedge i \geq 0$

Loop Body: Proof obligation

```
{ f * i! = n! /\ i >= 0 /\ i > 0 }
  f := f * i;
  i := i - 1;
{ f * i! = n! /\ i >= 0}
```

From Annotated Programs to Proofs

Loop Body

```
{ f * i! = n! /\ i >= 0 /\ i > 0 }      -- proof obligation
                                           for loop body
  f := f * i;
{ f * (i-1)! = n! /\ i >= 0 /\ i > 0}   -- new annotation!
  i := i-1;
{ f * i! = n! /\ i >= 0 }               -- end loop body
```

6. $\{f * (i-1)! = n! \wedge (i-1) \geq 0\}$ i := i − 1 $\{f * i! = n! \wedge i \geq 0\}$(Assignment)
7. $\{(f * i) * (i-1)! = n! \wedge (i-1) \geq 0\}$ f := f * i $\{f * (i-1)! = n! \wedge (i-1) \geq 0\}$ (Assignment)
8. $f * i! = n! \wedge i \geq 0 \wedge i > 0 \to (f * i) * (i-1)! = n! \wedge (i-1) \geq 0$ (Logic)
9. $\{f * i! = n! \wedge i \geq 0 \wedge i > 0\}$ f := f * i $\{f * (i-1)! = n! \wedge (i-1) \geq 0\}$ (Prec. Stren., 7,8)
10. $\{f * i! = n! \wedge i \geq 0 \wedge i > 0\}$ f := f * i; i := i − 1 $\{f * i! = n! \wedge i \geq 0\}$ (Sequence, 6,9)

From Annotated Programs to Proofs

Loop Body to While Loop


```
{ f * i! = n! /\ i >= 0 }                -- premise of while
                                            rule: "I"
  while (i > 0) do
    { f * i! = n! /\ i >= 0 /\ i > 0 }  -- "I /\ b"
    f := f * i;
    i := i-1;
    { f * i! = n! /\ i >= 0 }            -- "I"
{ f * i! = n! /\ i >= 0 /\ not(i > 0) } -- conclusion of while
                                            "I /\ not b"
```


11. $\{f * i! = n! \wedge i \geq 0\}$
    `while (i > 0) do { f:= f*i; i:= i-1}`
    $\{f * i! = n! \wedge i \geq 0 \wedge \neg(i > 0)\}$ (While, 10)

Putting it all together

```
{ n >= 0 }
  f := 1;
  i := n;
{ f = 1 /\ i = n /\ n >= 0 }                           -- have already
==>                                                    -- postcond weakening
{ f * i! = n! /\ i >= 0 }
  while (i > 0) do
      f := f * i;
      i := i-1;
{ f * i! = n! /\ i >= 0 /\ not(i > 0) }   -- have already
==>                                                    -- postcond weakening
{ f = n! }
```

12. $f = 1 \wedge i = n \wedge n \geq 0 \rightarrow f * i! = n! \wedge i \geq 0$ (Logic)
13. $\{n \geq 0\}$ `f := 1; i := n` $\{f * i! = n! \wedge i \geq 0\}$ (Postcond. Weak., 5, 12)
14. $\{n \geq 0\}$ `program` $\{f * i! = n! \wedge i \geq 0 \wedge \neg(i > 0)\}$ (Seq., 13, 11)
15. $f * i! = n! \wedge i \geq 0 \wedge \neg(i > 0) \rightarrow f = n!$ (Logic)
16. $\{n \geq 0\}$ `program` $\{f = n!\}$ (Postcondition Weakening, 14, 15)

Hoare Logic: Total Correctness

Motto. Total Correctness = partial correctness + termination

New Notation.

$$[P]\ \mathtt{S}\ [Q]$$

- P and Q are precondition and postcondition, as before
- S is a code fragment

Meaning. *If* the precondition holds, then executing S *will terminate*, and the postcondition is true.

Example.

- $[P]$ $\mathtt{S}$ $[true]$ – S always terminates from precondition P
- $\{P\}$ $\mathtt{S}$ $\{false\}$ – S never terminates from precondition P

Rules for Total Correctness

Q. What are the rules for *total* correctness?

- assignment
- sequencing
- conditional
- pre/post strengthening/weakening

still work, as there's no danger of non-termination.

Problematic Rule. while (may introduce non-termination)

Assignment, revisited

$$\frac{}{[Q(e)]\ \texttt{x} \ := \ \texttt{e}\ [Q(x)]}$$

Assumptions. (fine for our toy language)

- evaluation of expression e terminates.
- evaluation of *e* *returns a number* (no exception e.g.)

In General.

- the expression can be recursively defined
- there may be errors, e.g. division by zero

28 / 49

Rules for Total Correctness

$$\frac{P_s \rightarrow P_w \qquad [P_w]\ \mathtt{S}\ [Q]}{[P_s]\ \mathtt{S}\ [Q]} \quad \text{(Precondition Strengthening)}$$

$$\frac{[P]\ \mathtt{S}\ [Q_s] \qquad Q_s \rightarrow Q_w}{[P]\ \mathtt{S}\ [Q_w]} \quad \text{(Postcondition Weakening)}$$

$$\frac{[P]\ \mathtt{S}_1\ [Q] \qquad [Q]\ \mathtt{S}_2\ [R]}{[P]\ \mathtt{S}_1 ; \mathtt{S}_2\ [R]} \quad \text{(Sequence)}$$

$$\frac{[P \wedge b]\ \mathtt{S}_1\ [Q] \qquad [P \wedge \neg b]\ \mathtt{S}_2\ [Q]}{[P]\ \mathtt{if\ b\ then}\ \mathtt{S}_1\ \mathtt{else}\ \mathtt{S}_2\ [Q]} \quad \text{(Conditional)}$$

Assumption. Evaluation of b always terminates (OK here)

Termination of Loops

Example


```
[y > 0]

while (y < r) do
  r := r - y;
  q := q + 1

[true]
```

Termination of Loops

Example

```
[y > 0]

while (y < r) do
  r := r - y;
  q := q + 1

[true]
```


Observations

- `q := q + 1` irrelevant
- `y` doesn't change, always positive
- `r` strictly decreases in each iteration
- `y < r` will eventually be false.

Termination of Loops: General Condition

Example

```
[y > 0]

while (y < r) do
  r := r - y;
  q := q + 1

[true]
```


Termination follows if we have a *variant* E that is,

- $E \geq 0$ at the beginning of each iteration
- E strictly decreases at each iteration

Q. What could be a variant in this example?

While Rule for Total Correctness

Goal. Show that

$$[P] \; \texttt{while b do S} \; [Q]$$

In Addition to partial correctness (e.g. finding I), find variant E such that

- $E \geq 0$ at the beginning of each iteration:

$$I \wedge b \rightarrow E \geq 0$$

- E is strictly decreasing in each iteration

$$[I \wedge b \wedge (E = n)] \; \texttt{S} \; [I \wedge E < n]$$

where n is an auxiliary variable not appearing elsewhere, to "remember" initial value of E

While Rule for Total Correctness

$$\frac{I \wedge b \rightarrow E \geq 0 \qquad [I \wedge b \wedge E = n]\ \mathtt{S}\ [I \wedge E < n]}{[I]\ \mathtt{while\ b\ do\ S}\ [I \wedge \neg b]}$$

where n is an auxiliary variable *not appearing elsewhere*.

Intuition.

- E is an upper bound to the number of loop iterations
- termination of functional programs: measure decrease in recursive call

Example

Goal. $[y > 0]$ `while (y < r) do r := r - y; q := q+1` $[\mathit{true}]$

Focus. Loop body `r := r - y; q := q + 1`

- want some invariant: let's just call it I
- want some variant: here r is decreasing

First Goal. The variant is ≥ 0 when we enter the loop:

- formally: $I \land (y < r) \to r \geq 0$
- this suggests $I \equiv y > 0$

Second Goal. The invariant is re-established, and the variant decreases

- formally:
  $[(y > 0) \land (y < r) \land r = n]$ `r := r = y; q:= q+1` $[(y > 0) \land r < n]$
- this seems to be right, so let's prove it!

Example Proof

Goal.

$[(y > 0) \wedge (y < r) \wedge r = n]$ `r := r - y; q:= q+1` $[(y > 0) \wedge r < n]$

First Assignment.

$$[(y > 0) \wedge (r < n)]\ \mathtt{q} := \mathtt{q} + 1\ [y > 0 \wedge r < n]$$

Second Assignment.

$$[(y > 0) \wedge (r - y < n)]\ \mathtt{r\ :=\ r{-}y}\ [y > 0 \wedge r < n]$$

Sequencing.

$$[(y > 0) \wedge (r - y < n)]\ \mathtt{r\ :=\ r{-}y;\ q\ :=\ q{+}1}\ [y > 0 \wedge r < n]$$

Precondition Strengthening.

$$[(y > 0) \wedge (y < r) \wedge (r = n)]\ \mathtt{r\ :=\ r\ -\ y;\ q\ :=\ q{+}1}\ [y > 0 \wedge r < n]$$

Completing the Proof

While Rule.

$$\frac{I \wedge b \rightarrow E \geq 0 \qquad [I \wedge b \wedge E = n]\ \mathtt{S}\ [I \wedge E < n]}{[I]\ \mathtt{while\ b\ do\ S}\ [I \wedge \neg b]}$$

1. $[(y > 0) \wedge (r < n)]$ $\mathtt{q} := \mathtt{q} + 1$ $[y > 0 \wedge r < n]$ (Assignment)
2. $[(y > 0) \wedge (r - y < n)]$ `r := r-y` $[y > 0 \wedge r < n]$ (Assignment)
3. $[(y > 0) \wedge (r - y < n)]$ `r := r-y; q := q+1` $[y > 0 \wedge r < n]$ (Sequence, 2, 1)
4. $(y > 0) \wedge (y < r) \wedge (r = n) \rightarrow (y > 0) \wedge (r - y < n)$ (Logic)
5. $[(y > 0) \wedge (y < r) \wedge (r = n)]$ `r := r-y; q := q+1` $[y > 0 \wedge r < n]$ (Prec. Streng., 2, 3)
6. $(y > 0) \wedge (y < r) \rightarrow r \geq 0$ (Logic.)
7. $[y > 0]$ `while (y < r) do r:= r-y; q:= q+1` $[y > 0 \wedge y \geq r]$ (While Loop, 5, 6)
8. $y > 0 \wedge y \geq r \rightarrow \mathit{true}$
9. $[y > 0]$ `while (y < r) do r:= r-y; q:= q+1` $[\mathit{true}]$ (Postc. Weak., 7, 8)

Second Example

```
[n >= 0]
  fact := 1;
  i := n;
  while (i > 0) do
    fact := fact * i;
    i := i - 1
[fact = n!]
```


Q1. What is the invariant (linking *n*, *fact* and *i*)?

```
before: fact = 1, i = n
1st iteration: fact = n, i = n-1
2nd iteration: fact = n * (n-1), i = n-2
    ...
last iteration: fact = n * ... * 1, i = 0
```

Invariant: $fact * i! = n!$

Second Example

```
[n >= 0]
  fact := 1;
  i := n;
  while (i > 0) do
    fact := fact * i;
    i := i - 1
[fact = n!]
```


Q2. Is the invariant $fact * i! = n!$ good enough?

- true initially: for $fact = 1$ and $i = n$.
- implies postcondition: $fact * i! = n! \wedge \neg(i > 0) \rightarrow fact = n!$

Stronger Invariant. $fact * i! = n! \wedge i \geq 0$

Second Example

```
[n >= 0]
  fact := 1;
  i := n;
  while (i > 0) do
    fact := fact * i;
    i := i - 1
[fact = n!]
```


Q3. What's the variant?

- i ≥ 0 for every iteration ($I \wedge b \rightarrow E \geq 0$)
- decreases with every iteration

Variant. $E \equiv i$

Proof Skeleton

Simple Assignments.

```
[n >= 0]
  fact := 1;
  i := n;
[n >= 0 /\ fact = 1 /\ i = n]
```


Applying While


```
[ fact * i! = n! /\ i >= 0 ]
  while (i > 0) do
    fact := fact * i;
    i := i - 1
[ fact * i! = n! /\ i >= 0 /\ i <= 0]
```


Missing Glue. Weakening / Strengthening

- from postcondition of assignments to precondition of while
- from postcondition of while to goal statement ($fact = n!$)

Zooming in on While

```
[ fact * i! = n! /\ i >= 0 ]
  while (i > 0) do
    fact := fact * i;
    i := i - 1
[ fact * i! = n! /\ i >= 0 /\ i <= 0]
```


Loop Body (Invariant: $fact * i! = n! \wedge i \geq 0$, Variant: i)

```
  [fact * i! = n! /\ i >= 0 /\ i > 0 /\ i = a]
    fact := fact * i;
    i := i - 1
[ fact * i! = n! /\ i >= 0 /\ i < a]
```


Positivity Condition. $fact * i! = n! \wedge i \geq 0 \wedge i > 0 \rightarrow i \geq 0$ (Maths)

While Rule: Soundness

$$\frac{I \land b \to E \geq 0 \qquad [I \land b \land E = n]\ \mathtt{S}\ [I \land E < n]}{[I]\ \mathtt{while\ b\ do\ S}\ [I \land \neg b]}$$

Partial Correctness.

- premises of the rule imply those of the rule for partial correctness
- so *if* the loop terminates, the postcondition holds

Missing. Termination

Termination Analysis

$$\frac{I \wedge b \to E \geq 0 \qquad [I \wedge b \wedge E = n]\ \mathtt{S}\ [I \wedge E < n]}{[I]\ \mathtt{while}\ b\ \mathtt{do}\ \mathtt{S}\ [I \wedge \neg b]}$$

Let σ be a state that validates the precondition I. If b is false in σ, we are done. Assume that b is true in σ, hence the value of E in σ is ≥ 0.

Induction on the value $\sigma(E)$ of E in state σ.
Base case. $\sigma(E) = 0$.

- Right premise implies that $E < 0$ after one iteration
- With left premise, get that $\neg b$ after one iteration
- hence the loop terminates after one iteration.

Step Case. Let $\sigma(E) = k + 1$

- after one iteration, have $\sigma(E) \leq k$
- statement follows by induction hypothesis.

Variation: More Expressive Logic

So far. In triples $\{P\}$ S $\{Q\}$ we have

- S a program fragment (we keep this for now)
- P and Q *propositional* formulae made from basic arithmetic

Q. How about expressing the following?

$$\{true\}\ \mathtt{x} := 2 * \mathtt{x}\ \{\text{even}(x)\}$$

A. We could say that $\text{even}(x) = \exists y. 2 * y = x$

Change. Allowing pre/postconditions to be *first order* formulae.

Example.

$$\{true\}\ \mathtt{x} := 2 * \mathtt{x}\ \{\exists y. 2 * y = x\}$$

Using the assignment axiom.

$$\{\exists y. 2 * y = 2 * x\}\ \mathtt{x} := 2 * \mathtt{x}\ \{\exists y. 2 * y = x\}$$

More Expressive Logic. Assertions are first-order formulae

- all rules remain valid
- Hoare-logic is (almost) insensitive to underlying logic

Variation: More Expressive Programs

Example Feature. Arrays

- allow expressions to contain $a[i]$
- (we assume that the index is always in scope)

Maximum Finding

```
m := a[0]
i := 1;
while (i < n) do
  if a[i] > m then m := a[i] else m := m;
  i := i + 1
```


Q. How do we express that m is the maximum array element?

A. Use first order logic.

- m is largest: $\forall k. 0 \leq k < n \to m \geq a[k]$
- m is in array: $\exists k. 0 \leq k < n \wedge m = a[k]$

Annotated Code

```
{n >= 1}
  m := a[0]
  i := 1;
{m = a[0] /\ i = 1 /\ n >= 1 }
==>
{forall k. 0 <= k < i -> m >= a[k] /\ i <= n}
  while (i < n) do
    if a[i] > m then m := a[i] else m := m;
    i := i + 1
{forall k. 0 <= k < i -> m >= a[k] /\ i <= n /\ i >= n}
==>
{forall k. 0 <= k < n -> m <= a[k]}
```


Invariant. $I \equiv \forall k.0 \leq k < i \rightarrow m \geq a[k] \wedge i \leq n$

- initially: $m = a[0] \wedge i = 1 \rightarrow I$
- at end: $I \wedge i \geq n \rightarrow \forall k.0 \leq k < n \rightarrow m \leq a[i]$

Remark. Can turn this into a formal proof as before.

References

The textbook has material on Hoare Logic

- Grassman & Tremblay, "*Logic and Discrete Mathematics: A Computer Science Perspective*", Prentice-Hall, Chapter 9, pp. 481-518.

Some nice online notes with lots of examples:

- Gordon, "*Specification and Verification I*", `http://www.cl.cam.ac.uk/~mjcg/Lectures/SpecVer1/SpecVer1.html`, Chapters 1-2, pp. 7-46.

A comprehensive early history of Hoare Logic appears in

- Apt, K.R., *Ten Years of Hoare Logic: A Survey*", ACM Transactions on Programming Languages and Systems, October, 1981.