

# Hoare Logic III

**COMP2600 — Formal Methods for Software Engineering**

Ranald Clouston

Australian National University

Semester 2, 2011

## Proof Rule for While Loops (Rule 6/6)

$$\frac{\{P \wedge b\} S \{P\}}{\{P\} \text{ while } b \text{ do } S \{P \wedge \neg b\}}$$

- $P$  is called the **loop invariant**.
- $P$  is true before we encounter the while statement, and remains true each time around the loop (although not necessarily *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.

## Validating the While Rule

Assume  $P$  holds in the initial state.

If  $b$  is false immediately then the program state is not changed, so  $P \wedge \neg b$  holds.

Now assume the invariant holds:  $\{P \wedge b\} S \{P\}$ .

$P$  therefore holds at the end of each iteration.

Assume that the loop terminates, and so  $b$  eventually becomes false. Then (as above) the program state does not change from there, so  $P \wedge \neg b$  holds.

***(What about non-termination?)***

## Example

Suppose we want to find a precondition  $P$  such that:

$$\{P\} \text{ while } (n > 0) \text{ do } n := n - 1 \{n = 0\}$$

We want a loop invariant  $P$  such that

- if  $P$  is true initially,  $P$  remains true each time around the loop;
- $(P \wedge \neg(n > 0)) \leftrightarrow (n = 0)$

So the following looks a reasonable loop invariant:

$$P \equiv (n \geq 0)$$

The premise of the while rule then follows from the assignment axiom:

$$\{(n \geq 0) \wedge (n > 0)\} n := n - 1 \{(n \geq 0)\}$$

## 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  $n^2$ )**

## A Very Informal Analysis

Let's look at some examples:

$$1 = 1 = 1^2$$

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

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

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

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

## How can we prove it?

First we need a loop invariant  $P$ .

Think about the postcondition and the control condition:

```
while (i ≠ n) do
  i:=i+1;
  s:=s+(2*i-1)
{ $s = n^2$ }
```

From the while rule, we want  $(P \wedge i = n) \leftrightarrow (s = n^2)$

Think about the loop body. Each time,  $i$  increments and  $s$  moves on to the next square number.

**Loop invariant  $P \equiv (s = i^2)$  seems plausible.**

## Check $P$ is an invariant:

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

1.  $\{s + (2 * i - 1) = i^2\} \mathbf{s := s + (2 * i - 1)} \{s = i^2\}$  (Assignment)
2.  $\{s + (2 * (i + 1) - 1) = (i + 1)^2\} \mathbf{i := i + 1} \{s + (2 * i - 1) = i^2\}$  (Asst.)
3.  $\{s = i^2\} \mathbf{i := i + 1} \{s + (2 * i - 1) = i^2\}$  (Simplify)
4.  $\{s = i^2\} \mathbf{i := i + 1; s := s + (2 * i - 1)} \{s = i^2\}$  (Sequencing 1,3)

So far, so good. ( $P$  **is an invariant**.)

## Completing the Proof

1. Strengthen the precondition to match the While rule premise:

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

2. Now we can apply the While rule:

$$\{s = i^2\} \mathbf{while \dots s:=s+(2*i-1)} \{s = i^2 \wedge i = n\}$$

3. Check that the **initialisation establishes the invariant:**

$$\{0 = 0^2\} \mathbf{i:=0; s:=0} \{s = i^2\}$$

4.  $(0 = 0^2) \equiv \mathit{True}$ , so putting it all together with Sequencing we have

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

as required.

## The Proof Process in Review

- Coming up with an invariant requires *intuition* and may not be obvious!
- We've been looking for  $P$  such that

$$(P \wedge \neg b) \leftrightarrow Q$$

where  $Q$  is the desired postcondition. In fact, because of *postcondition weakening* all we need is

$$P \wedge \neg b \rightarrow Q$$

- In our first example

$$\{P\} \text{ while } (n > 0) \text{ do } n := n - 1 \{n = 0\}$$

There are various options for  $P$ , e.g.  $(n = 0)$  or *False*. The one we used,  $(n \geq 0)$ , is 'better' because it is weaker.

## What about Termination?

Remember that Hoare Logic is for *Partial Correctness*.

Consider the code `while True do x:=0`. This will loop forever!

However you can still prove things about it, e.g.

$$\{True\} \text{ while True do } x:=0 \{False\}$$

(left as an exercise).

There are separate techniques to show termination. For example, one technique is to define a **loop variant** that changes every time we go around the loop but cannot change forever (e.g. a decreasing natural number).

## Are the Rules Complete?

We have defined rules for a very simple language; new rules would be needed if we wanted e.g. `for` loops, functions...

But are our rules the ‘right’ ones for our little language?

- We have focused on Soundness (i.e. every provable Hoare triple is true).
- Although we showed that each of the rules is sound, there are some assumptions we haven’t really discussed.
- With the same assumptions, the rules are also complete<sup>a</sup> for the language.

---

<sup>a</sup> A logic is complete if every true expression is provable in the logic

## 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 variable.)

How is *aliasing* a problem?

- Suppose  $x$  and  $y$  refer to the same cell of memory.
  - We get  $\{y + 1 = 5 \wedge y = 5\} \mathbf{x:=y+1} \{x = 5 \wedge y = 5\}$
  - i.e.  $\{y = 4 \wedge y = 5\} \mathbf{x:=y+1} \{x = 5 \wedge y = 5\}$

## 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/~mjc/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.

***Remember - public holiday next Monday!***