

# Correctness Triples

## CS 536 Lecture 7, Mon Feb 1, 2012

### A. Why

- Correctness triples are how we write a program with its specification.
- Proof rules for correctness triples will show us how to reason about programs and their specifications.

### B. Objectives

- At the end of today you should
- Be able to understand the intuitive meaning of a given triple.
- Be able to prove that a simple assignment triple is correct using the "backward" assignment rule.

### C. Correctness Triples

- To review, a correctness triple  $\{p\} S \{q\}$  is valid if it's satisfied in all states.
- Under partial correctness,  $\{p\} S \{q\}$  is satisfied in a state  $\sigma$  if  $\sigma$  satisfies  $p$  and  $M(S, \sigma)$  exists implies  $M(S, \sigma)$  satisfies  $q$
- Under total correctness,  $\{p\} S \{q\}$  is satisfied in  $\sigma$  if  $\sigma$  satisfies  $p$  implies  $M(S, \sigma)$  exists and satisfies  $q$
- $M(S, \sigma)$  exists if  $S$  terminates under  $\sigma$  (i.e., when executed in state  $\sigma$ )
  - It doesn't exist if  $S$  diverges under  $\sigma$  (goes into an infinite loop when started in  $\sigma$ ).

### D. More Correctness Triple Examples

- **Example 1:**  $\{x = 0\} \neq \{x \geq 0\} x := x - 1 \{x \geq 0\}$ , so the triple isn't valid  
(Program has a bug!)
- **Example 2:**  $\models \{x > 0\} x := x - 1 \{x \geq 0\}$  (Fix bug by strengthening precondition)
- **Example 3:**  $\models \{x - 1 \geq 0\} x := x - 1 \{x \geq 0\}$  (Another way to phrase Example 2)
- **Example 4:**  $\models \{x \geq 0\} x := x - 1 \{x \geq -1\}$  (Fix bug by weakening postcondition)
- **Example 5:**  $\models \{x = x_0 \geq 0\} x := x - 1 \{x_0 \geq 0 \wedge x = x_0 - 1\}$   
(Another way to phrase Example 4:  $x_0$  is "the value of  $x$  before the assignment")
- **Example 6:**  $\{x \geq 0\} \text{ if } x > 0 \text{ then } x := x - 1 \text{ fi } \{x \geq 0\}$   
(Fix bug by changing the program)

In Examples 7 – 10, let  $W \equiv \mathbf{while\ } x > 0 \ \mathbf{do\ } x := x-1 \ \mathbf{od}$

- **Example 7:**  $\models_{tot} \{\mathbf{true}\} W \{x \leq 0\}$   
(If we start (in a state) with  $x > 0$ , we run until  $x = 0$ ; if we start with  $x \leq 0$ , we stop immediately.)
- **Example 8:**  $\models_{tot} \{x \geq 0\} W \{x \leq 0\}$   
(From Example 7 and  $\mathbf{true} \leftrightarrow x \geq 0 \vee x < 0$ , this triple is just a special case of  $\models_{tot} \{x \geq 0 \vee x < 0\} W \{x \leq 0\}$ .)
- **Example 9:**  $\models_{tot} \{x \geq 0\} W \{x = 0\}$   
(This makes a stronger (more precise) claim than Example 8.)
- **Example 10:**  $\models \{x = x_0\} W \{(x_0 \geq 0 \rightarrow x = 0) \wedge (x_0 < 0 \rightarrow x = x_0)\}$   
(Here,  $x_0$  stands for "the value of  $x$  before the program ran," so in English, the postcondition says "If we start with  $x \geq 0$ , then  $x = 0$  on termination; otherwise,  $x$  is unchanged." This is like Example 7, but the postcondition is more precise.)

In Examples 11 – 14, let  $U \equiv \mathbf{while\ } x \neq 0 \ \mathbf{do\ } x := x-1 \ \mathbf{od}$

- **Example 11:**  $\models_{tot} \{x \geq 0\} U \{x = 0\}$   
(If we start with  $x$  positive, then  $U$  does terminate, and with  $x = \text{zero}$ .)
- **Example 12:**  $\models \{x \geq 0\} U \{x = 0\}$   
(If a program is totally correct, it's also partially correct.)
- **Example 13:**  $\models \{x < 0\} U \{x = 0\}$   
(**IF**  $U$  terminates (i.e., if false) then  $x$  is zero.)
- **Example 14:**  $\not\models_{tot} \{x < 0\} U \{x = 0\}$  (i.e., exists  $\sigma$  such that  $\sigma \not\models_{tot} \{x < 0\} U \{x = 0\}$ )  
(It is not the case that in all states in which  $x < 0$ ,  $U$  terminates with  $x = 0$ ; in fact, the triple is satisfied in none of the states where  $x < 0$ .)

Here are some more assignment examples. Note: Integer division truncates.

- **Example 15:**  $\{m/2 > 0\} m := m/2 \{m > 0\}$   
(Actually, this is true whether division truncates or rounds or whatever.)
- **Example 16:**  $\{(m = 2*n \vee m = 2*n+1) \wedge n > 0\} m := m/2 \{m > 0\}$   
(Like Example 14 but we've given the name  $n$  to the value of  $m/2$ .)
- **Example 17:**  

$$\{ \mathbf{left} < \mathbf{right} \wedge \mathbf{left} \neq \mathbf{right}-1 \}$$

$$\mathbf{mid} := (\mathbf{left} + \mathbf{right}) / 2$$

$$\{ \mathbf{left} < \mathbf{mid} < \mathbf{right} \}$$

(In binary search, if  $\mathbf{left}$  and  $\mathbf{right}$  are at least 2 apart, their midpoint is strictly between them.)

- **Example 18:**

```
{ left < mid < right ∧ right-left = d }
right := mid
{ right-left < d }
```

(In binary search, moving the right endpoint to the middle reduces the distance between the endpoints.)

### **Reasoning About Assignments (Technique 1: The Backward Rule)**

- There are two general rules for reasoning about assignments; here's one of them.
- If  $P(x)$  is a predicate function, then in general  $\{P(e)\} v := e \{P(v)\}$

- **Example 19:** If  $P(x) \equiv x > 0$  then

- $\{m/2 > 0\} m := m/2 \{m > 0\}$
- $\{P(m/2)\} m := m/2 \{P(m)\}$

- **Example 20:** If  $P(x) \equiv x - \text{left} < d$  then

- $\{\text{mid} - \text{left} < d\} \text{right} := \text{mid} \{\text{right} - \text{left} < d\}$
- $\{P(\text{mid})\} \text{right} := \text{mid} \{P(\text{right})\}$
- Note: If we're given  $\text{left} < \text{mid} < \text{right} \wedge \text{right} - \text{left} = d$ , we can infer that  $\text{mid} - \text{left} < d$ , so it should follow that
 
$$\{\text{left} < \text{mid} < \text{right} \wedge \text{right} - \text{left} = d\} \text{right} := \text{mid} \{\text{right} - \text{left} < d\}$$
 (which was Example 18 above).

- **Example 21:** If  $P(x) \equiv s = \text{sum}(0, x)$  then

- $\{s = \text{sum}(0, k+1)\} k := k+1 \{s = \text{sum}(0, k)\}$
- $\{P(k+1)\} k := k+1 \{P(k)\}$

- **Example 22:** If  $P(x) \equiv x = \text{sum}(0, k+1)$  then

- $\{s+(k+1) = \text{sum}(0, k+1)\} s := s+(k+1) \{s = \text{sum}(0, k+1)\}$
- $\{P(s+(k+1))\} s := s+(k+1) \{P(s)\}$

- For the assignment rule  $\{P(e)\} v := e \{P(v)\}$  to be valid, we need the following lemma
  - **Assignment Lemma:** For all  $\sigma$ , if  $\sigma \models P(e)$  then  $M(v := e, \sigma) = \sigma[v \mapsto \sigma(e)] \models P(v)$ .
  - We won't go into a detailed proof of this lemma, but hopefully it makes sense intuitively: If  $P(v)$  means "We know property  $P$  about  $v$ ", and we're assigning  $v$  the value  $\sigma(e)$ , then we need to know property  $P$  about  $e$  before the assignment, but that's exactly what  $\sigma \models P(e)$  means.