

Class 08: More Assignment, More Loops

A. Why?

Studying more examples will help with understanding the proof rules and with using proof outlines.

B. Outcomes

By the end of the class you should

- Understand the different versions of the assignment rule and when to use them.
- Be familiar with loops with conditionals in their body (e.g. binary search).

C. Midterm Exam (60 min)

D. Derived Assignment Rules

- The regular assignment rule is “backward”: $\{p[u := t]\} u := t \{p\}$
- It’s possible to derive assignment rules that work “forward”.

Forward Assignment Rule 1

$\{p \wedge u = c\} u := t \{p[u := c] \wedge u = t[u := c]\}$ where c is a constant

- (Let’s make substitution have high precedence.)
- c stands for the value of u before the assignment.
- Probably a named constant that doesn’t appear in the program, just the proof.

An example: $\{x > 0 \wedge x = c\} x := x + 1 \{c > 0 \wedge x = c + 1\}$

Another example:

$\{0 \leq i \leq n \wedge s = \text{sum}(0, i) \wedge i = i'\}$

$i := i + 1$

$\{0 \leq i' \leq n \wedge s = \text{sum}(0, i') \wedge i = i' + 1\}$

$\{0 \leq i' \leq n \wedge s = \text{sum}(0, i') \wedge i = i' + 1 \wedge s = s'\}$

$s := s + i$

$\{0 \leq i' \leq n \wedge s' = \text{sum}(0, i') \wedge i = i' + 1 \wedge s = s' + i\}$

- Derivation from backward assignment rule:

$\{p \wedge u = c\}$

$\{p[u := c] \wedge t = t[u := c]\}$

$\{p[u := c][u := t] \wedge u[u := t] = t[u := c][u := t]\}$

$\{(p[u := c] \wedge u = t[u := c])[u := t]\}$

$u := t$

$\{p[u := c] \wedge u = t[u := c]\}$

Forward Assignment Rule 2

- If u is a new variable, we can simplify even more:

$$\{p\} u := t \{p \wedge u = t\} \text{ where } u \text{ not free in } p \text{ or } t.$$

- Derivation from forward assignment rule 1:

$$\{p\} \{p \wedge u = c\} u := t \{p[u := c] \wedge u = t[u := c]\} \{p \wedge u = t\}$$

- Some explanation:

- For the first implication, we introduce c as a new named constant so that $p \rightarrow p \wedge u = c$ is the definition of c .
- For the last implication, u is not free in p or t , so $p[u := c] \equiv p$ and $t[u := c] \equiv t$.

Example:

$$\{n \geq 0\} i := 0; \{n \geq 0 \wedge i = 0\} s := 0 \{n \geq 0 \wedge i = 0 \wedge s = 0\}$$

$$\{0 \leq i \leq n \wedge s = \text{sum}(0, i)\}$$

We've been using

$$\{n \geq 0\}$$

$$\{0 \leq 0 \leq n \wedge 0 = \text{sum}(0, 0)\}$$

$$i := 0;$$

$$\{0 \leq i \leq n \wedge 0 = \text{sum}(0, i)\}$$

$$s := 0$$

$$\{0 \leq i \leq n \wedge s = \text{sum}(0, i)\}$$

E. Binary Search Example (Version 1)

- An example that uses a forward assignment rule is binary search: calculating the midpoint introduces it as a new variable.
- Let $\text{Sorted}(b, n) \equiv \forall i: 0 \leq i < n \rightarrow b[i] \leq b[i+1]$.
- Specification:

$$\{\text{Sorted}(b, n) \wedge n > 0 \wedge b[0] \leq x < b[n]\}$$

$$\text{binsearch}(b, x, n)$$

$$\{0 \leq lt < n \wedge b[lt] \leq x < b[lt+1]\}$$

- (It's handy to have $b[n]$ be a sentinel value.)
- Let invariant $p \equiv 0 \leq lt < rt \leq n \wedge b[lt] \leq x < b[rt]$
 - Treat b, n as named constants, so $\text{Sorted}(b, n)$ can be used anywhere, so it doesn't have to be part of p .
- Initialization $\{p[lt := 0][rt := n]\} lt := 0; rt := n \{p\}$
 $p[lt := 0][rt := n] \equiv 0 \leq 0 < n \leq n \wedge b[0] \leq x < b[n]$. (Hence the outer precondition.)
- Loop test: Loop until $rt = lt + 1$; loop while $rt \neq lt + 1$
- Loop so far:

```

{q0} lt := 0 ; rt := n ;
{inv p} while rt ≠ lt+1 do ... od
{p ∧ rt = lt+1}
{q1}

```

where $q_0 \equiv \text{Sorted}(b, n) \wedge n \geq 1 \wedge b[0] \leq x < b[n]$

$p \equiv 0 \leq lt < rt \leq n \wedge b[lt] \leq x < b[rt]$

$q_1 \equiv 0 \leq lt < n \wedge b[lt] \leq x < b[lt+1]$

- For the loop body, we calculate a midpoint $m := (lt+rt)/2$ (with truncating division) and make progress via $lt := m$ or $rt := m$.
- What needs to be true before making progress?

- Check $\{p[lt := m]\} \text{ } lt := m \text{ } \{p\}$

$$p[lt := m] \equiv (0 \leq lt < rt \leq n \wedge b[lt] \leq x < b[rt])[lt := m]$$

$$\equiv 0 \leq m < rt \leq n \wedge b[m] \leq x < b[rt]$$

- Check $\{p[rt := m]\} \text{ } rt := m \text{ } \{p\}$

$$p[rt := m] \equiv (0 \leq lt < rt \leq n \wedge b[lt] \leq x < b[rt])[rt := m]$$

$$\equiv 0 \leq lt < m \leq n \wedge b[lt] \leq x < b[m]$$

- Establishing whether $b[m] \leq x$ or $x < b[m]$ can be done with an if test.

- What about $lt < m < rt$?

- Together $lt < rt$ and $rt \neq lt+1$ imply $rt - lt \geq 2$. Turns out that's enough.

- So the loop body is

```
{p ∧ rt ≠ lt+1}
```

```
m := (lt+rt)/2;
```

```
{p ∧ rt ≠ lt+1 ∧ m = (lt+rt)/2} /* call this p1 */
```

```
if b[m] ≤ x then
```

```
  {p1 ∧ b[m] ≤ x} {p[lt := m]} lt := m {p}
```

```
else
```

```
  {p1 ∧ b[m] > x} {p[rt := m]} rt := m {p}
```

```
fi {p}
```

- Minimal proof outline:

```
{q0 ≡ Sorted(b, n) ∧ n ≥ 1 ∧ b[0] ≤ x < b[n]}
```

```
lt := 0 ; rt := n ;
```

```
{inv p ≡ 0 ≤ lt < rt ≤ n ∧ b[lt] ≤ x < b[rt]}
```

```
while rt ≠ lt+1 do
```

```
  m := (lt+rt)/2;
```

```
  if b[m] ≤ x then lt := m else rt := m fi
```

```
od {q1 ≡ 0 ≤ lt < n ∧ b[lt] ≤ x < b[lt+1]}
```

F. Binary Search (Version 2)

- The search program above is a bit inefficient because it doesn't stop early if it finds x (partly because it finds the rightmost occurrence of x).
- Use new postcondition q_2 and invariant p_2 :
 - $q_2 \equiv 0 \leq lt < n \wedge (b[lt] = x \vee b[lt] < x < b[lt+1])$
 - $p_2 \equiv 0 \leq lt < rt \leq n \wedge (b[lt] = x \vee b[lt] < x < b[rt])$
 - Now if $b[m] = x$, we can set $lt = rt - 1 = m$ and halt immediately.
- The initialization code and loop test are unchanged, but its correctness proof is different:

```

{q0 ≡ Sorted(b, n) ∧ n ≥ 1 ∧ b[0] ≤ x < b[n]} /* Same q0 */
{p2[rt:=n][lt:=0]} lt:=0; {p2[rt:=n]} rt:=n;
invariant p2 while rt ≠ lt+1 do ... od
{p2 ∧ rt = lt+1}
{q2 ≡ 0 ≤ lt < n ∧ (b[lt] = x ∨ b[lt] < x < b[lt+1])}

```

- The loop body changes because we test for $b[m] <$, $=$, or $> x$:

```

{p2 ∧ rt ≠ lt+1}
m := (lt+rt)/2;
{p2 ∧ rt ≠ lt+1 ∧ m = (lt+rt)/2} /* call this p3 */
if b[m] < x then
  {p3 ∧ b[m] < x} {p2[lt:=m]} lt:=m; {p2}
else {p3 ∧ b[m] ≥ x} if b[m] > x then
  {p3 ∧ b[m] ≥ x ∧ b[m] > x} {p2[rt:=m]} rt:=m {p2}
else {p3 ∧ b[m] ≥ x ∧ b[m] ≤ x}
  {p2[rt:=lt+1][lt:=m]} lt:=m; {p2[rt:=lt+1]}
  {p2[rt:=lt+1]} rt:=lt+1; {p2}
fi {p2} fi {p2}

```

- With just the code and outer conditions, we have the minimal proof outline for the loop body:

```

{p2 ∧ rt ≠ lt+1}
m := (lt+rt)/2;
if b[m] < x then
  lt:=m;
else if b[m] > x then
  rt:=m
else
  lt:=m; rt:=lt+1;
fi fi {p2}

```

- The minimal proof outline is

```

{ $q_0 \equiv \text{Sorted}(b, n) \wedge n \geq 1 \wedge b[0] \leq x < b[n]$ } /* Same  $q_0$  */
 $lt := 0$  ;  $rt := n$  ;
{inv  $p_2 \equiv 0 \leq lt < rt \leq n \wedge (b[lt] = x \vee b[lt] < x < b[rt])$ }
while  $rt \neq lt+1$  do
   $m := (lt+rt)/2$ ;
  if  $b[m] < x$  then  $lt := m$  else if  $b[m] > x$  then  $rt := m$ 
  else
     $lt := m$ ;  $rt := lt+1$ ;
  fi fi od
{ $q_2 \equiv 0 \leq lt < n \wedge (b[lt] = x \vee b[lt] < x < b[lt+1])$ }

```

G. Greatest Common Divisor

- For $a, b \in \mathbb{N}$, $a, b > 0$, $\text{gcd}(a, b)$ is the largest value that divides both a and b evenly (without remainder).
- E.g., $\text{gcd}(300, 180) = \text{gcd}(2^2 * 3 * 5^2, 2^2 * 3^2 * 5) = 2^2 * 3 * 5 = 60$.
- Useful gcd property:
 - Case 1 ($a > b$): $\text{gcd}(a, b) = \text{gcd}(a - b, b)$
 - Case 2 ($b > a$): $\text{gcd}(a, b) = \text{gcd}(a, b - a)$
 - Case 3 ($a = b$): $\text{gcd}(a, b) = a = b$
- E.g., $\text{gcd}(300, 180) = \text{gcd}(120, 180)$, $\text{gcd}(120, 60) = \text{gcd}(60, 60) = 60$.
- Here's an iterative gcd -calculating loop:

```

/*  $a'$  and  $b'$  are the values of  $a$  and  $b$  before the program runs. */
{ $a > 0 \wedge b > 0 \wedge a = a' \wedge b = b'$ }
{inv  $p \equiv a > 0 \wedge b > 0 \wedge \text{gcd}(a', b') = \text{gcd}(a, b)$ }
while  $a \neq b$  do
  if  $a > b$  then  $a := a - b$  else  $b := b - a$  fi
od
{ $a = \text{gcd}(a', b')$ }

```

- A full annotation:

```

{ $a > 0 \wedge b > 0 \wedge a = a' \wedge b = b'$ }
{inv  $p \equiv a > 0 \wedge b > 0 \wedge \text{gcd}(a', b') = \text{gcd}(a, b)$ }
while  $a \neq b$  do
  { $p \wedge a \neq b$ }
  if  $a > b$  then
    { $p \wedge a \neq b \wedge a > b$ } { $p[a := a - b]$ }  $a := a - b$  { $p$ }
  else

```

```

    {p ∧ a ≠ b ∧ a ≤ b} {p[b := b - a]} b := b - a {p}
  fi {p}
od
{p ∧ a = b}
{a = gcd(a', b')}
```