Lecture Summary: SE2011

March 25, 2015

1 Basic Hoare Logic

We began by reviewing some of the basic rules of Hoare logic:

$$\frac{\{\alpha\}\, P\, \{\beta\} \qquad \beta \Rightarrow \gamma}{\{\alpha\}\, P\, \{\gamma\}} \qquad \text{weaken postcondition}$$

$$\frac{\alpha \Rightarrow \beta \qquad \{\beta\}\, P\, \{\gamma\}}{\{\alpha\}\, P\, \{\gamma\}} \qquad \text{strengthen precondition}$$

$$\{\alpha[e/x]\}\, x := e\, \{\alpha\} \qquad \text{assignment}$$

$$\frac{\{\alpha\}\, P\, \{\beta\} \qquad \{\beta\}\, Q\, \{\gamma\}}{\{\alpha\}\, P; Q\, \{\gamma\}} \qquad \text{sequencing}$$

$$\frac{\{\alpha \wedge c\}\, P\, \{\beta\} \qquad \{\alpha \wedge \neg c\}\, Q\, \{\beta\}}{\{\alpha\}\ \mathbf{if}\, c\, \mathbf{then}\, P\, \mathbf{else}\, Q\, \{\beta\}} \qquad \text{if-then-else}$$

$$\frac{\{\alpha \wedge c\}\, P\, \{\beta\} \qquad (\alpha \wedge \neg c \Rightarrow \beta)}{\{\alpha\}\ \mathbf{if}\, c\, \mathbf{then}\, P\, \{\beta\}} \qquad \text{if-then}$$

$$\frac{\{\alpha \wedge c\}\, P\, \{\alpha\}}{\{\alpha\}\ \mathbf{while}\, c\, \mathbf{do}\, P\, \mathbf{od}\ \{\alpha \wedge \neg c\}} \qquad \text{while}$$

We then discussed some simple programs, and saw that these could be derived from their specifications.

2 Summing an array

Notation: $[0, n) = \{i \in \mathbf{N} \mid 0 \leq i < n\}$.

We are given an array $A[0, n)$ with indices from $[0, n)$. Here the specification is:

$$\{\mathbf{true}\}\, P\, \{s = \sum_{j \in [0,n)} A[j]\}$$

We guess the loop invariant $s = \sum_{j\in[0,i)} A[j]$, and obtain the following program, annotated with assertions $\{\ldots\}$ at various points:

$\{\mathbf{true}\}$
s:= 0;
i:= 0;
$\{s = \sum_{j\in[0,i)} A[j]\}$ (loop invariant)
while i $\neq$ n
do
  s := s + A[i];
  $\{s = \sum_{j\in[0,i+1)} A[j]\}$
  i:= i+1 ;
  $\{s = \sum_{j\in[0,i)} A[j]\}$
od
$\{s = \sum_{j\in[0,n)} A[j]\}$

To verify correctness of these annotations, we need to check the following basic statements:

- $\{\mathbf{true}\}$ s:= 0; i:= 0 $\{s = \sum_{j\in[0,i)} A[j]\}$

  By two applications of the assignment rule and sequencing, it suffices for this that $\mathbf{true} \Rightarrow 0 = \sum_{j\in[0,0)} A[j]$. This holds by the convention that an empty sum equals 0.

- $\{s = \sum_{j\in[0,i)} A[j] \wedge i \neq n\}$ s := s + A[i] $\{s = \sum_{j\in[0,i+1)} A[j]\}$

  We prove this by the assignment rule and the strengthen precondition rule. The assignment rule gives
  $\{s + A[i] = \sum_{j\in[0,i+1)} A[j]\}$ s := s + A[i] $\{s = \sum_{j\in[0,i+1)} A[j]\}$
  Simple mathematics gives
  $(s = \sum_{j\in[0,i)} A[j] \wedge i \neq n) \Rightarrow s + A[i] = \sum_{j\in[0,i+1)} A[j]$
  The conclusion then follows using the strengthen precondition rule.

- $\{s = \sum_{j\in[0,i+1)} A[j]\}$ i:= i+1 $\{s = \sum_{j\in[0,i)} A[j]\}$

  This is immediate using the assignment rule.

- Note that the previous two steps give, using sequencing, that
  $\{s = \sum_{j\in[0,i)} A[j] \wedge i \neq n\}$ s := s + A[i] ; i:= i+1 $\{s = \sum_{j\in[0,i)} A[j]\}$
  Using the while rule, we now conclude
  $\{s = \sum_{j\in[0,i)} A[j]\}$ **while** i $\neq$ n **do** s := s + A[i] ; i:= i+1 **od** $\{s = \sum_{j\in[0,i)} A[j] \wedge i = n\}$

- By simple logic, $s = \sum_{j\in[0,i)} A[j] \wedge i = n \Rightarrow s = \sum_{j\in[0,n)} A[j]$, so by the weakening the postcondition rule, we have
  $\{s = \sum_{j\in[0,i)} A[j]\}$ **while** i $\neq$ n **do** s := s + A[i] ; i:= i+1 **od** $\{s = \sum_{j\in[0,n)} A[j]\}$

- Combining this with the conclusion about the initialization section, using the sequencing rule, we obtain that the program as a whole satisfies its spec.

We discussed in class how this reasoning could be run backwards to derive the program pieces from their local specifications.

We also mentioned a subtlety (are the array accesses always safe?) that is actually a very important question from the point of view of security, but did not carefully develop a proof that shows this. This will be taken up in your mentor meeting.

3 Finding the rightmost element

The second example we considered was that of finding the rightmost occurrence of the value 1 in an array. We discussed a program that does this by starting at the right of the array and moves to the left until a 1 is found. We started to develop a specification and proof, but did not complete this. You will finish this example in Assignment 1b!