CS 4110 – Programming Languages and Logics
Lecture #11: Hoare Logic: Decorated Programs

1 Decorated Programs

Doing a complete proof in Hoare Logic can feel overly verbose. The "core" of a proof consists of the preconditions and postconditions surrounding every command; with those, it's usually possible to infer the complete shape of the proof tree.

Decorating programs is a way to informally write down a Hoare logic proof of correctness. The idea is to insert assertion decorations between every "line" of the program. Using this informal evidence, you can reconstruct the full formal proof.

1.1 Informal Rules

We'll set out a few rules to check whether a decorated program represents a valid proof using *local consistency* checks. The rules, as you'll see, are informal reflections of the formal inference rules for Hoare logic.

Skip. For **skip**, the precondition and postcondition should be the same, like this:

$$\begin{array}{l} \{P\} \\ \textbf{skip} \\ \{P\} \end{array}$$

Sequence. For sequences, a new assertion R appears between the two commands. The two halves $\{P\}\ c_1\ \{R\}$ and $\{R\}\ c_2\ \{Q\}$ must be (recursively) locally consistent:

$$\begin{array}{l} \{P\} \\ c_1; \\ \{R\} \\ c_2 \\ \{Q\} \end{array}$$

Assignment. Assignments are locally consistent when the precondition is the same as the postcondition except that it substitutes the assigned expression in for the variable:

$$\begin{array}{l} \{P[a/x]\} \\ x := a \\ \{P\} \end{array}$$

Conditions. An **if** is locally consistent when both branches are locally consistent after adding the branch condition to each:

$$
\begin{array}{l}
\{P\} \\
\textbf{if } b \textbf{ then} \\
\quad \{P \wedge b\} \\
\quad c_1 \\
\quad \{Q\} \\
\textbf{else} \\
\quad \{P \wedge \neg b\} \\
\quad c_2 \\
\quad \{Q\} \\
\{Q\}
\end{array}
$$

Loops. A **while** command should be decorated with a loop invariant:

$$
\begin{array}{l}
\{P\} \\
\textbf{while } b \textbf{ do} \\
\quad \{P \wedge b\} \\
\quad c \\
\quad \{P\} \\
\{P \wedge \neg b\}
\end{array}
$$

Implication. To capture the CONSEQUENCE rule, you can always write a (valid) implication to connect two commands:

$$
\begin{array}{l}
\{P\} \Rightarrow \\
\{Q\}
\end{array}
$$

1.2 Decorating a Program

These informal rules tell you how to *verify* whether a decorated program is correct, but they don't by themselves tell you how to *construct* those decorations (and thereby a proof). You can do this almost mechanically—except for loop invariants, which still require some creativity to concoct.

Here's an example program:

$$
\begin{array}{l}
\textbf{while } (0 < \mathsf{y}) \textbf{ do } (\\
\quad \mathsf{x} := \mathsf{x} + 1; \\
\quad \mathsf{y} := \mathsf{y} - 1 \\
)
\end{array}
$$

The first step is to decide what we want to prove about this program. So we write a precondition and postcondition for the whole program. Intuitively, the program adds y onto the initial value of the variable x, so we'll assert that:

$$
\begin{array}{l}
\{\mathsf{x} = m \ \wedge \ \mathsf{y} = n \ \wedge \ 0 \leq n\} \\
\textbf{while } (0 < \mathsf{y}) \textbf{ do } (\\
\quad \mathsf{x} := \mathsf{x} + 1; \\
\quad \mathsf{y} := \mathsf{y} - 1 \\
) \\
\{\mathsf{x} = m + n\}
\end{array}
$$

This program is a **while** command, so the next step is to come up with a loop invariant. We'll set up the structure first. We need an assertion I where we can "connect" the loop's precondition and postcondition to our overall precondition and postcondition using implications, like this:

$$
\begin{array}{l}
\{\mathsf{x} = m \ \wedge \ \mathsf{y} = n \ \wedge \ 0 \leq n\} \Rightarrow \\
\{I\} \\
\textbf{while } (0 < \mathsf{y}) \textbf{ do } (\\
\quad \{I \wedge 0 < y\} \\
\quad \mathsf{x} := \mathsf{x} + 1; \\
\quad \mathsf{y} := \mathsf{y} - 1 \\
\quad \{I\} \\
) \\
\{I \wedge 0 \not< \mathsf{y}\} \Rightarrow \\
\{\mathsf{x} = m + n\}
\end{array}
$$

On every iteration of the loop, the variable x is the sum we want, $m + n$, less the current value of y, which is the number of iterations remaining. So we'll define the invariant I like this:

$$I ::= (\mathsf{x} = m + n - \mathsf{y}) \wedge 0 \leq \mathsf{y}$$

The top implication follows because $n - \mathsf{y} = 0$, and the bottom implication is valid because $0 \not< y$ and $0 \leq y$ together imply $y = 0$.

To finish decorating the program, we need an assertion between the two lines in the body of the loop. By the rule for assignments, we know that this must be $I[\mathsf{y} - 1/\mathsf{y}]$, or:

$$P_1 ::= (\mathsf{x} = m + n - (\mathsf{y} - 1)) \wedge (0 \leq \mathsf{y} - 1)$$

To make the assignment to x locally consistent, then, its precondition must be $P_1[\mathsf{x} + 1/\mathsf{x}]$, or:

$$P_2 ::= (\mathsf{x} + 1 = m + n - (\mathsf{y} - 1)) \wedge (0 \leq \mathsf{y} - 1)$$

It's straightforward to see that the precondition at the top of the loop, $I \wedge 0 < y$, implies this new assertion P_2. Now we can write our complete decorated program using these definitions:

$\{\mathsf{x} = m \ \wedge \ \mathsf{y} = n \ \wedge \ 0 \leq n\} \Rightarrow$
$\{I\}$
while $(0 < \mathsf{y})$ **do** (
$\quad\{I \wedge 0 < y\} \Rightarrow$
$\quad\{P_2\}$
$\quad\mathsf{x} := \mathsf{x} + 1;$
$\quad\{P_1\}$
$\quad\mathsf{y} := \mathsf{y} - 1$
$\quad\{I\}$
)
$\{I \wedge 0 \not< \mathsf{y}\} \Rightarrow$
$\{\mathsf{x} = m + n\}$