Lecture 18

Assignments

Recall the meaning of \( \langle P \rangle \ C \ \langle Q \rangle \), partial and total correctness, and rules in Lecture 17.

Three steps in doing a proof of partial correctness for assignments and conditionals:

1. First **annotate** using the appropriate inference rules.

2. Then work **backwards for assignments** in the proof.

3. Finally **prove any “implied”s:**
   - Annotations from (1) above containing implications
   - Adjacent assertions created in step (2).

Proofs here are written in Math 135 style unless specified otherwise. They can use Predicate Logic, basic arithmetic, or any other appropriate reasoning.

Example: Show following is satisfied under partial correctness.

\[
\langle x = x_0 \land y = y_0 \rangle \\
t = x; \\
x = y; \\
y = t; \\
\langle x = y_0 \land y = x_0 \rangle
\]

This results in the following annotated program:

\[
\langle x = x_0 \land y = y_0 \rangle \\
\langle y = y_0 \land x = x_0 \rangle \quad \text{implied} \quad \langle P_3[x/t]\rangle \quad \text{[proof required]} \\
t = x; \\
\langle y = y_0 \land t = x_0 \rangle \quad \text{assignment} \quad P_3 \text{ is } \langle P_2[y/x]\rangle \\
x = y; \\
\langle x = y_0 \land t = x_0 \rangle \quad \text{assignment} \quad P_2 \text{ is } \langle P[t/y]\rangle \\
y = t; \\
\langle x = y_0 \land (y = x_0) \rangle \quad \text{assignment} \quad \langle P \rangle
\]

Finally, show \( \langle x = x_0 \land y = y_0 \rangle \) implies \( \langle y = y_0 \land x = x_0 \rangle \). This is clear.
Conditionals

if-then-else:
\[
\frac{\langle P \land B \rangle \ C_1 \ \langle Q \rangle \ \langle P \land \neg B \rangle \ C_2 \ \langle Q \rangle}{\langle P \rangle \ \text{if } (B) \ C_1 \ \text{else} \ C_2 \ \langle Q \rangle} \quad \text{(if-then-else)}
\]

if-then (without else):
\[
\frac{\langle P \land B \rangle \ C \ \langle Q \rangle \ (P \land \neg B) \rightarrow Q}{\langle P \rangle \ \text{if } (B) \ C \ \langle Q \rangle} \quad \text{(if-then)}
\]

Note: B is the boolean statement for the if-then(-else) clause.

Annotated program template for if-then:
\[
\begin{array}{l}
\langle P \rangle \\
\text{if (B) } \\
\langle P \land B \rangle \quad \text{if-then} \\
\quad C \langle Q \rangle \\
\quad \quad \text{[add justification based on } C]\end{array}
\]
\[
\langle Q \rangle \quad \text{if-then implied: proof of } (P \land \neg B) \rightarrow Q
\]

Annotated program template for if-then-else:
\[
\begin{array}{l}
\langle P \rangle \\
\text{if (B) } \\
\langle P \land B \rangle \quad \text{if-then-else} \\
\quad C_1 \langle Q \rangle \\
\quad \quad \quad \text{(justify depending on } C_1 \text{—a “subproof”)}
\end{array}
\]
\[
\begin{array}{l}
\text{else } \\
\langle P \land \neg B \rangle \quad \text{if-then-else} \\
\quad C_2 \langle Q \rangle \\
\quad \quad \text{(justify depending on } C_2 \text{—a “subproof”)}
\end{array}
\]
\[
\langle Q \rangle \quad \text{if-then-else [justifies this } Q, \text{ given previous two]}
\]

Example: Prove the following code on the left is satisfied under partial correctness.
\[
\begin{array}{l}
\langle \text{true} \rangle \\
\text{if ( max < x ) } \{ \\
\quad \text{max = x ;} \\
\quad \} \\
\langle max \geq x \rangle \\
\end{array}
\]
\[
\begin{array}{l}
\langle P \rangle \\
\text{if ( B ) } \{ \\
\quad C \\
\quad \} \\
\langle Q \rangle
\end{array}
\]
This code is clearly an application of the if-then rule, as illustrated on the right side. Using the annotated if-then template, we can fill in the necessary justifications.

```java
if ( max < x )
    (\(true \land max < x\)) \(\rightarrow\) if-then
    max = x ;
    (\(Q\)) \(???
}

(\(Q\)) \(\rightarrow\) if-then implied: Proof of \((true \land \neg(max < x)) \rightarrow Q\)
(\(max \geq x\))

Now we can see that \(Q\) should be justified via the assignment rule. The assignment rule states that if a predicate holds where an expression \(E\), in this case is \(x\), replaces all the free occurrences of \(max\), then after \(max=x\), we can have the unsubstituted version of the predicate—but there is a free occurrence of \(max\) above it, so we need to push up for the assignment and not have \(max\) occur at all in the annotation prior to the assignment (in this case we should restrict to just \(x\)).

This means we need to introduce an implied statement. Looking at our goal, we see that if we had the line \(x \geq x\), then we can get \(max \geq x\) via assignment.

```java
if ( max < x )
    (\(true \land max < x\)) \(\rightarrow\) if-then
    \(x \geq x\) \(\rightarrow\) implied\(_1\): Proof of \((true \land (max < x)) \rightarrow x \geq x\)
    max = x ;
    (\(max \geq x\)) \(\rightarrow\) assignment
}
```

(\(max \geq x\)) \(\rightarrow\) if-then implied\(_2\): Proof of \((true \land \neg(max < x)) \rightarrow max \geq x\)

Now all that is left is to prove the implied statements...

Implied\(_1\): Proof of \((true \land (max < x)) \rightarrow x \geq x\)
Clearly \(x \geq x\) is a tautology (from the equality as being defined as the conjunction of both \(\leq\) and \(\geq\) and thus the implication holds).

Implied\(_2\): Proof of \((true \land \neg(max < x)) \rightarrow max \geq x\)

Exercise. (or see Common Overheads).

For additional examples of if-then(-else) annotated programs, see the Common Overheads.
**While Loops**

Proving Total Correctness means proving both Partial Correctness and Program Termination. Programs containing loops may not terminate (i.e. infinite loop). To start, we will focus on only the partial correctness of programs with loops and later move into showing total correctness.

**partial-while**:

\[
\frac{\langle I \land B \rangle \ C \ \langle I \rangle}{\langle I \rangle \ \text{while} \ (B) \ C \ \langle I \land \neg B \rangle} \quad \text{(partial-while)}
\]

Condition \( I \) is called a **loop invariant**. Invariant means never changing.

Annotated program template for **partial-while**:

\[
\langle P \rangle \\
\langle I \rangle \quad \text{implied (a)} \\
\text{while} \ (B) \ \{ \\
\langle I \land B \rangle \quad \text{partial-while} \\
C \\
\langle I \rangle \quad \leftarrow \text{to be justified, based on C} \\
\} \\
\langle I \land \neg B \rangle \quad \text{partial-while} \\
\langle Q \rangle \quad \text{implied (b)}
\]

As you can see, the loop invariant, holds throughout the template. It is never changing.

But where does \( I \) come from? Well... we need to determine \( I \) ourselves!

A **loop invariant** is an assertion that is true both *before* and *after* each execution of the body of a loop. An invariant may or may not be useful in proving termination (to discuss later).

**Example**: Find a loop invariant for the following code.

It is helpful to trace the states of the program at the while statement.

Since we do not know the value of \( x \), let’s choose \( x = 5 \).

\[
\langle x \geq 0 \rangle \\
\text{y = 1 ;} \\
\text{z = 0 ;} \\
\rightarrow \text{while (z != x) \{ } \\
\text{z = z + 1 ;} \\
\text{y = y * z ;} \\
\} \\
\langle y = x! \rangle \\
\]

State at **while** statement:

<table>
<thead>
<tr>
<th>state</th>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
<th>( z \neq x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 )</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>true</td>
</tr>
<tr>
<td>( s_4 )</td>
<td>5</td>
<td>24</td>
<td>4</td>
<td>true</td>
</tr>
<tr>
<td>( s_5 )</td>
<td>5</td>
<td>120</td>
<td>5</td>
<td>false</td>
</tr>
</tbody>
</table>

Now we can use the trace to find a statement that holds throughout the table.

In addition, we need \( \langle I \land (\neg(z = x)) \rangle \) to imply \( \langle y = x! \rangle \) and \( \langle x \geq 0 \rangle \) to imply \( \langle I \rangle \).

This means that choosing \( y \geq z \) is unhelpful as it does not help with the implication.

The statement we choose is known as the **candidate loop invariant**.
This was left for you to think about over the weekend.
We will return to this example on Tuesday.