Starter Questions

Feel free to discuss these with your neighbour:

- Do you remember Natural Deduction proofs from the Maths course? How did you construct those?
- What is the difference between the inference system for semantics and the inference system for correctness that I have given you?
- How many loop invariants can a loop have?
- If I find a loop invariant for a loop does that mean I can prove the loop correct?
- How many loop invariants do I need if I have 3 loops? What if one of those loops is nested inside another?
- Did we need $y > 0$ to prove the division algorithm partially correct?

You will be given both inference systems in the Exam

The Exercises for Monday are Online
Lecture 5
Total Correctness and Practical Correctness
COMP11212

Giles Reger

March 2017
Axiomatic System

\[
\text{ass}_p \quad \{ P[x \mapsto A[a]] \} \ x := a \{ P \} \\
\text{skip}_p \quad \{ P \} \text{ skip } \{ P \} \\
\text{comp}_p \quad \{ P \} S_1 \{ Q \}, \{ Q \} S_2 \{ R \} \\
\quad \{ P \} S_1; S_2 \{ R \} \\
\text{if}_p \quad \{ P \land B[b] \} S_1 \{ Q \}, \{ P \land \neg B[b] \} S_2 \{ Q \} \\
\quad \{ P \} \text{ if } b \text{ then } S_1 \text{ else } S_2 \{ Q \} \\
\text{while}_p \quad \{ P \land B[b] \} S \{ P \} \\
\quad \{ P \} \text{ while } b \text{ do } S \{ P \land \neg B[b] \}
The rule for \texttt{while} loops is

\[
\{ P \land B[b] \} \ S \ \{ P \} \\
\} \ \texttt{while} \ b \ \texttt{do} \ S \ \{ P \land \neg B[b] \}
\]

where \( P \) is referred to as a \textit{loop invariant}.

We motivated this informally before (last week). Remember, what is true at the end of one iteration will be true at the start of the next.
Does this program always terminate?

\[ \text{if } x > y \text{ then } z := x \text{ else } z := y \]
Does this program always terminate?

\textbf{if } x > y \textbf{ then } z := x \textbf{ else } z := y

Programs without loops will always result in a state in a finite number of rewritings using $\Rightarrow$. We can prove this as a property of the programming language.
Does this program always terminate?

\[ \begin{align*}
\text{r} & := x; \\
\text{d} & := 0; \\
\text{while } y \leq r \text{ do } (d := d + 1; \ r := r - y)
\end{align*} \]
Does this program always terminate?

\[ r := x; \]
\[ d := 0; \]
\[ \textbf{while} \ y \leq r \ \textbf{do} \ (d := d+1; \ r := r - y) \]

Why? When does it terminate?
Does this program always terminate?

\[
\begin{align*}
\text{r} & := \text{x} ; \\
\text{d} & := 0 ; \\
\text{while } \text{y} \leq \text{r} \text{ do } ( \text{d} := \text{d+1}; \ \text{r} := \text{r} - \text{y} )
\end{align*}
\]

Why? When does it terminate? When \( \text{y} > 0 \)
Does this program always terminate?

\[
\begin{align*}
r &:= x; \\
d &:= 0; \\
\textbf{while } y \leq r \textbf{ do } (d := d+1; \ r := r-y)
\end{align*}
\]

Why? When does it terminate? When \( y > 0 \)

If \( y > 0 \) then the value of \( r \) decreases every time we go round the loop.

The loop terminates when \( y > r \) i.e. when \( r \) becomes smaller than \( y \) (as the value of \( y \) isn’t changing).

Whatever the size of \( r \) at the start, we can only decrease it a finite number of times before it is less than \( y \).
Total Correctness: The Formal Bit

Only one thing changes, the rule for while

\[
\frac{[P \land B[b] \land E = n] \quad C \quad [P \land E < n]}{[P] \text{ while } b \text{ do } C \quad [P \land \neg B[b]]} \quad \text{if } P \land B[b] \rightarrow E \geq 0
\]

Here \( E \) is a loop variant i.e. an expression that decreases on every iteration. It is important to note that the loop condition needs to bound this expression from below.

It is not guaranteed that we can always find a loop variant for a terminating loop.

Conceptually, \( E \) is the maximum number of loop iterations left
def div(x, y):
    assert x>=0 and y>0
    r=x
    d=0
    while y <= r:
        assert x===(d*y)+r and r>=0
        d=d+1
        r=r-y
    assert x===(d*y)+r and r>=0 and r<y
    return (d, r)
public static Result div(int x, int y) {
    assert (x >= 0 && y > 0) : "Precondition Fails";
    int r = x;
    int d = 0;
    while (y <= r) {
        assert (x == (d * y) + r &&
                r >= 0) : "Loop Invariant Fail";
        d = d + 1;
        r = r - y;
    }
    assert (x == (d * y) + r &&
            r >= 0 && r < y) : "Postcondition Fails"
    return new Result(d, r);
}
Assertions can act as extra documentation
Some tools can statically prove assertions
Some tools can use assertions for automated test generation
Some research (mine) considers temporal assertions
Some research (in my group) considers the problem of automatically generating loop invariants
Some research considers the termination problem in general