Do you remember?

- We specify programs using Hoare triples \( \{ P \} S \{ Q \} \)
- We apply the assignment rule \( \{ P[x \mapsto A[a]] \} x := a \{ P \} \)

  backwards e.g. start with \( P \) and \( S \) and generate a precondition

- The rule of consequence is used to glue proofs together after other rules have been applied
- To reason about loops we use loop invariants

Something to think about:

- Do these loops always terminate? How can we show this?

  ```
  while x > 0 do x := x - 1; y := y + 1
  ```

  ```
  while x < 100 do x := x + 1; y := y - 1
  ```

  ```
  r := x; d := 0; while y \leq r do (d := d + 1; r := r - y)
  ```
Admin Stuff

The Unit Survey is available on Blackboard

- We really value your feedback
- It holds a lot more weight when there’s more (last year 12%)

Reminder about Exam

- Single hybrid exam (online and paper together)
- Two online sections (Part 1 and 2) worth 15 marks each
- Two paper sections (Part 1 and 2) worth 15 marks each

Comment about Coursework (online and examples classes):

- Exam questions similar in style to what you’ve seen
- Many coursework questions are harder than exam (Part 2)
- Look at Solutions for Revision (online and examples)
1. Apply the rules to decompose into smaller triples until get a sequence of assignments \( \{ P \} \ A_1; \ldots; A_n \ \{ Q \} \)

2. If program contains if statements then we get multiple triples

3. Apply assignment rule backwards to get a new precondition \( Q' \) e.g.

\[ \{ Q' \} \ A_1 \ \{ Q_1 \} \ \ldots \ \{ Q_{n-1} \} \ A_n \ \{ Q \} \]

4. Try and prove implications \( P \rightarrow Q' \) (this is the glue)
Pseudo-Algorithm For Correctness When Just Loop

Assume triple of form \( \{ P \} \) while \( b \) do \( S \) \( \{ Q \} \)

1. Find a loop invariant \( I \) such that \( P \rightarrow I \) and \( (I \land \neg b) \rightarrow Q \)
   
   - Make sure these implications hold before spending time on next step

2. Prove \( \{ I \land b \} S \{ I \} \) by previous approach, iterating this one, or using next one
For triples \{ P \} S_1; \ldots; S_n \{ Q \} where some of \( S_i \) are while loops

1. For each loop \( S_i \) find a loop invariant \( I_i \)
2. Make a high-level sequence of composed triples e.g.

\[
\{ P \} S_1 \{ R_1 \} \\
\ldots \\
\{ R_{n-1} \} S_n \{ R_n = Q \}
\]

if \( S_i \) is a loop then \( R_{i-1} = R_i \) will be loop invariant \( I_i \)
3. Prove each triple by previous approaches or by iterating this one
Axiomatic System

\[
\begin{align*}
\text{ass}_p & : \{ P[x \mapsto A[a]] \} x := a \{ P \} \\
\text{skip}_p & : \{ P \} \text{skip} \{ P \} \\
\text{if}_p & : \begin{cases} 
\{ P \land B[b] \} S_1 \{ Q \}, \\
\{ P \land \neg B[b] \} S_2 \{ Q \}
\end{cases} \\
& \quad \{ P \} \text{if } b \text{ then } S_1 \text{ else } S_2 \{ Q \} \\
\text{while}_p & : \begin{cases} 
\{ P \land B[b] \} S \{ P \} 
\end{cases} \\
& \quad \{ P \} \text{while } b \text{ do } S \{ P \land \neg B[b] \} \\
\text{cons}_p & : \begin{cases} 
\{ P' \} S \{ Q' \} 
\end{cases} \\
& \quad \begin{cases} 
\{ P \} S \{ Q \}
\end{cases} \quad \text{if } P \Rightarrow P' \text{ and } Q' \Rightarrow Q
\end{align*}
\]
Worked Examples

{ x > y } x := 0; y := 0 { x = y }

{ x = 2y } if x > 0 then x := x + 1 { x > 0 \rightarrow \text{odd}(x) }

{ x \geq 0 \land x = a } y := 1; \text{while } x > 0 \text{ do } (y := 3y; x := x - 1) \{ y = 3^x \}

{ x = a \land y = b } \text{while } x \neq 0 \text{ do } (x := x - 1; y := y + 1) \{ y = a + b \}
Worked Examples

\[
\{ x > y \} \ x := 0; \ y := 0 \{ x = y \}
\]

\[
\{ x = 2y \} \text{ if } x > 0 \text{ then } x := x + 1 \{ x > 0 \rightarrow \text{odd}(x) \}
\]

\[
\{ x \geq 0 \land x = a \} \ y := 1; \text{while } x > 0 \text{ do } (y := 3y; x := x - 1) \{ y = 3^x \}
\]

\[
[ x = a \land y = b ] \text{ while } x \neq 0 \text{ do } (x := x - 1; y := y + 1) [ y = a + b ]
\]
Does this program always terminate?

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

\[ \text{if } x > y \text{ then } z := x \text{ else } z := y \]

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?

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

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

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

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

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 \( y \).
Only one thing changes, the rule for while

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

\[
\text{if } P \land B[b] \to E \geq 0
\]

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 and r<y) : "Postcondition Fails"
    return new Result(d, r);
}

Need to run java -ea Program to run with assertions
• 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