Example

\[
\vdash \{ A[x \to e] \} x := e \{ A \}
\]

\[
\vdash \{ A \land b \} c_1 \{ B \} \quad \vdash \{ A \land \neg b \} c_2 \{ B \}
\]

\[
\vdash \{ A \} \text{if } b \text{ then } c_1 \text{ else } c_2 \{ B \}
\]

\[
\vdash A' \Rightarrow A \vdash \{ A \} c \{ B \} \vdash B \Rightarrow B'
\]

\[
\vdash \{ A' \} c \{ B' \}
\]

\[
\vdash \{ A \land b \} c \{ A \}
\]

\[
\vdash \{ A \} \text{while } b \text{ do } c \{ A \land \neg b \}
\]

\[
\vdash \{ A \} c_1 \{ C \} \quad \vdash \{ C \} c_2 \{ B \}
\]

\[
\vdash \{ A \} c_1 ; c_2 \{ B \}
\]

\[
\{ x=x_0 \land y=y_0 \}
\]

if (x > y) {
    t = x - y;
    while (t > 0) {
        x = x - 1;
        y = y + 1;
        t = t - 1;
    }
}

\[
\{ x_0 > y_0 \Rightarrow y=x_0 \land x=y_0 \}
\]
Today

• How to prove program termination
• Pragmatics of axiomatic-style verification
  • Human effort
• Formula solver performance
  - e.g., based on sizes of proof obligations we generate
From partial to total correctness

Square brackets indicate total correctness, where we prove termination!

\[
[A \land b] c_1 [B] \quad [A \land \neg b] c_2 [B]
\]

\[
[A] \text{ if } b \text{ then } c_1 \text{ else } c_2 [B]
\]

\[
[A] c_1 [C] \quad [C] c_2 [B]
\]

\[
[A] c_1 ; c_2 [B]
\]

What about loops?
Rank function

Function F of the state that
- a) Maps state to an integer
- b) Decreases with every iteration of the loop
- c) Is guaranteed to stay greater than zero
- Also called **variant** function

\[
[A \land b \land F = z] \land c [A \land F < z] \quad (A \land b \Rightarrow F \geq 0)
\]

[A] while b do c [A \land \neg b]
Can we prove this?

\[
[x = x_0 \land y = y_0]
\]

if \((x > y)\) {
  \(t = x - y;\)
  while \((t > 0)\) {
    \(x = x - 1;\)
    \(y = y + 1;\)
    \(t = t - 1;\)
  }
}

\[
[x_0 > y_0 \Rightarrow y = x_0 \land x = y_0]
\]

The main insight we need:
Loop variant
\(F = t\)
Soundness

We gave a semantic soundness condition for \( \{A\} \ c \ \{B\} \):
\[
\forall \sigma, \sigma'. (A(\sigma) \land (\sigma, c) \downarrow \sigma') \Rightarrow B(\sigma')
\]

What condition captures the intent of \([A] \ c \ [B]\)?
(1) \(\forall \sigma, \sigma'. (A(\sigma) \land (\sigma, c) \downarrow \sigma') \Rightarrow B(\sigma')\); and
(2) \(\forall \sigma. A(\sigma) \Rightarrow \exists \sigma'. (\sigma, c) \downarrow \sigma' \) ("terminates whenever \(A\")

See Coq code for this lecture for a proof that our judgment satisfies this soundness condition.
Constructing Hoare logic proofs manually is tedious. We should be able to automate most of it.
Weakest Preconditions

Weakest predicate P such that \{P\} \implies \{A\}.
- P weaker than Q iff Q \implies P

\[
wp(\text{skip}, A) = A
\]
\[
wp(x := e, A) = A[e/x]
\]
\[
wp(c_1; c_2, A) = wp(c_1, wp(c_2, A))
\]
\[
wp(\text{if } b \text{ then } c_1 \text{ else } c_2, A) = (b \land wp(c_1, A)) \lor (\neg b \land wp(c_2, A))
\]
Weakest Precondition

“while” is tricky!

Let $W = \text{wp}(\text{while } e \text{ do } c, A)$
Then: $W \leftrightarrow (e \Rightarrow \text{wp}(c, W) \land \neg e \Rightarrow A)$

[A recursive equation, where it isn't obvious a solution exists!]

Pragmatic solution: ask programmers to annotate loops with loop invariants.

c := x := e | c; c | if b then c else c | \{I\} while b do c
Adding the “while” case

\[
\begin{align*}
wp(x := e, A) &= A[e/x] \\
wp(c_1 ; c_2, A) &= wp(c_1, wp(c_2, A)) \\
wp(\text{if } b \text{ then } c_1 \text{ else } c_2, A) &= (b \land wp(c_1, A)) \lor (\neg b \land wp(c_2, A)) \\
wp(\{I\} \text{ while } b \text{ do } c, A) &= I \\
&\quad \land \text{written}(c) = \{x_1, \ldots, x_n\} \\
&\quad \land (\forall x_1, \ldots, x_n. I \land b \Rightarrow wp(c, I)) \\
&\quad \land (\forall x_1, \ldots, x_n. I \land \neg b \Rightarrow A)
\end{align*}
\]
Theorem (Completeness of this verification scheme):

For any command $c$ and postcondition $B$, there exists a command $c'$ annotated with proper loop invariants, such that for any candidate precondition $A$, if $\models \{A\} \; c \; \{B\}$, then $A \Rightarrow \text{wp}(c', B)$.

Proof.

By induction on $c$.

Trickiest case: “while” (unsurprisingly) Need to pick a good loop invariant for arbitrary “while $b$ do $c$” and $B$. This one works:

$$\lambda \sigma. \forall \sigma'. (\sigma, \text{while } b \text{ do } c) \downarrow \sigma' \Rightarrow B(\sigma')$$

It's a bit sneaky, referring to the semantics of “while” directly!

(And it's ill-formed in normal first-order logic, but fine in Coq.)
Performance considerations

Can we get exponential blow-up in formula size via wp?

Yes!

if (b) { x := y } else { x := z };  
if (b) { x := y } else { x := z };  
   ...
if (b) { x := y } else { x := z }

For $n$ sequential “if”s, 
the postcondition is repeated $2^n$ times!

One solution: get the programmer to help 
by adding **invariant assertions** in code.
Assertions

c := x := e | c; c | if b then c else c | {l} while b do c | assert(A)

“assert” can act like “skip” in the operational semantics! It's only here as a proof hint.

wp(assert(A), B) = A ∧ (∀σ. A(σ) ⇒ B(σ))

Improved program:

if (b) { x := y } else { x := z }; assert(x > 0 ∧ y > 0 ∧ z > 0);
if (b) { x := y } else { x := z }; assert(x > 0 ∧ y > 0 ∧ z > 0);
...
if (b) { x := y } else { x := z }
Another performance issue

Let \( c = \text{if } (b) \{ x := y \} \text{ else } \{ x := z \}; \text{ assert}(x > 0 \land y > 0 \land z > 0) \)

\[
wp(c, A) = \\
(b \land y > 0 \land y > 0 \land z > 0 \\
\land (\forall x, y, z. x > 0 \land y > 0 \land z > 0 \Rightarrow A)) \\
\lor (\neg b \land z > 0 \land y > 0 \land z > 0 \\
\land (\forall x, y, z. x > 0 \land y > 0 \land z > 0 \Rightarrow A))
\]

A state-independent formula is duplicated!
We should be able to lift those formulas to the top level.
Define two separate notions:
\(wp(c, A) = \) what must be true before running \(c\) to ensure \(A\)
\(vc(c, A) = \) state-independent facts needed to support \(wp\)

**Theorem**: \(vc(c, A) \Rightarrow \{wp(c, A)\} \ c \ \{A\}.\)

\(wp(x := e, A) = A[e/x]\)
\(wp(c_1; c_2, A) = wp(c_1, wp(c_2, A))\)
\(wp(\text{if } b \text{ then } c_1 \text{ else } c_2, A) = (b \land wp(c_1, A)) \lor (\neg b \land wp(c_2, A))\)
\(wp(\{l\} \text{ while } b \text{ do } c, A) = l\)

\(vc(x := e, A) = \top\)
\(vc(c_1; c_2, A) = vc(c_2, A) \land vc(c_1, wp(c_2, A))\)
\(vc(\text{if } b \text{ then } c_1 \text{ else } c_2, A) = vc(c_1, A) \land vc(c_2, A)\)
\(vc(\{l\} \text{ while } b \text{ do } c, A) = vc(c, I) \land \text{written}(c) = \{x_1, \ldots, x_n\}\)
\(\land (\forall x_1, \ldots, x_n. I \land b \Rightarrow wp(c, I))\)
\(\land (\forall x_1, \ldots, x_n. I \land \neg b \Rightarrow A)\)
Coq demo

See L13-Verification.v for:
• Performance comparisons of these methods across small examples
• Formalization & proof of all the ideas from this lecture