Formal Correctness Proofs and Proof Outlines
CS 536: Science of Programming, Fall 2018

A. Why
- A formal proof lets us write out in detail the reasons for believing that something is valid.
- Proof outlines give us a way to show the same information as a proof, but in an easier-to-use form.

B. Objectives
At the end of this lecture you should
- Know how the invariant, initialization, test, body, and postcondition of a loop are interrelated.
- Know how to write a formal proof of correctness.
- Recognize a full proof outline and translate it to a formal proof of correctness.
- Translate between a full proof outline and a minimal proof outline.

C. An Example of a Loop Invariant
- **Example 1**: Here’s a simple loop program that calculates \( s = \text{sum}(0, n) = 0+1+\ldots+n \) where \( n \geq 0 \). (If \( n < 0 \), define \( \text{sum}(0, n) = 0 \).) Note the loop invariant appears explicitly as a comment.

```
{ n \geq 0 }
i := 0; s := 0;
{ \text{inv } p_1 \equiv 0 \leq i \leq n \land s = \text{sum}(0, i) } 
while i < n do
  s := s+i+1; i := i+1
od 
{ s = \text{sum}(0, n) }
```
- Informally, to see that this program works, we need
  - \{ n \geq 0 \} i := 0; s := 0 \{ p_1 = 0 \leq i \leq n \land s = \text{sum}(0, i) \}
  - \{ p_1 \land i < n \} s := s+i+1; i := i+1 \{ p_1 \}
  - \{ p_1 \land i \geq n \rightarrow s = \text{sum}(0, n) \}
- It’s straightforward to use wp or sp to show that the two triples are correct. A bit of predicate logic gives us the implication, which we need to weaken the loop’s postcondition to the one we want.
  - We’ll do a detailed analysis in a little while.

D. Alternative Invariants Yield Different Programs and Proofs
- The invariant, test, initialization code, and body of a loop are all interconnected: Changing one can change them all. For example, we use \( s = \text{sum}(0, i) \) in our invariant, so we have the loop terminate with \( i = n \).
  - If instead we use \( s = \text{sum}(0, i+1) \) or \( s = \text{sum}(0, i-1) \) in our invariant, we must terminate with \( i+1 = n \) or \( i-1 = n \) respectively, and we change what we should increment \( s \) by.
• Example 1: Using $s = \text{sum}(0, i)$

\[
\begin{align*}
\{ n \geq 0 \} \\
i := 0; s := 0; \\
\{ \text{inv } p_1 = 0 \leq i \leq n \land s = \text{sum}(0, i) \} \\
\text{while } i < n \text{ do} \\
\quad s := s+i+1; i := i+1 \\
\text{od} \\
\{ s = \text{sum}(0, n) \}
\end{align*}
\]

• Example 2: Using $s = \text{sum}(0, i+1)$

\[
\begin{align*}
\{ n > 0 \} \\
i := 0; s := 1; \\
\{ \text{inv } p_2 = 0 \leq i < n \land s = \text{sum}(0, i+1) \} \\
\text{while } i < n-1 \text{ do} \\
\quad s := s+i+2; i := i+1 \\
\text{od} \\
\{ s = \text{sum}(0, n) \}
\end{align*}
\]

• Example 3: Using $s = \text{sum}(0, i-1)$

\[
\begin{align*}
\{ n \geq 1 \} \\
i := 1; s := 0; \\
\{ \text{inv } p_2 = 1 \leq i \leq n+1 \land s = \text{sum}(0, i-1) \} \\
\text{while } i \leq n \text{ do} \\
\quad s := s+i; i := i+1 \\
\text{od} \\
\{ s = \text{sum}(0, n) \}
\end{align*}
\]

E. Formal Correctness Proofs

• A formal proof of correctness specifies the triples we're claiming to be valid and the proof rules that let us justify those claims.
• They're very rigid syntactically compared to the informal discussions of correctness we typically have.
• The difference between a formal proof and informal proof is like the difference between a program written in a programming language versus an algorithm.
• There are different ways to write out formal proofs. The simplest is the Hilbert-style proof (you may have seen it in high-school geometry). It consists of a list of lines; each line contains a judgement and a justification.
• Each line’s assertion is an assumption, an axiom, or follows by some rule that appeals to earlier lines in the proof.
1. Length of $AB = \text{length of } XY$ \hspace{1cm} \text{Assumption}
2. Angle $ABC = \text{Angle } XYZ$ \hspace{1cm} \text{Assumption}
3. Length of $BC = \text{length of } YZ$ \hspace{1cm} \text{Assumption}
4. Triangles $ABC, XYZ$ are congruent \hspace{1cm} \text{Side-Angle-Side, lines 1, 2, 3}

**F. Sample Formal Proof**

- We can write out the reasoning for the sample summation loop we looked at.
- **Example 1 (repeated):**

  
  {\begin{align*}
  &\{n \geq 0\} \\
  &i := 0; s := 0; \\
  &\{\text{inv } p_1 = 0 \leq i \leq n \land s = \text{sum}(0, i)\}
  
  &\text{while } i < n \text{ do} \\
  &\hspace{1cm} s := s + i + 1; i := i + 1 \\
  &\text{od} \\
  &\{s = \text{sum}(0, n)\}
  \end{align*}}

- In the formal proof below, let $S_1 \equiv s := s + i + 1; i := i + 1$ (the loop body) and let $W \equiv \text{while } i < n \text{ do } S_1 \text{ od }$ (the loop). Recall $p_1 = 0 \leq i \leq n \land s = \text{sum}(0, i)$.

  \begin{align*}
  &1 \quad \{n \geq 0\} \ i := 0 \ \{n \geq 0 \land i = 0\} \quad \text{Assignment} \\
  &2 \quad \{n \geq 0 \land i = 0\} \ s := 0 \ \{n \geq 0 \land i = 0 \land s = 0\} \quad \text{Assignment} \\
  &3 \quad \{n \geq 0\} \ i := 0; s := 0 \ \{n \geq 0 \land i = 0 \land s = 0\} \quad \text{Sequence, lines 1, 2} \\
  &4 \quad n \geq 0 \land i = 0 \land s = 0 \rightarrow p_1 \quad \text{Predicate logic} \\
  &5 \quad \{n \geq 0\} \ i := 0; s := 0 \ \{p_1\} \quad \text{Postcond. weak., 3, 4} \\
  &6 \quad \{p_1[i+1/i]\} i := i + 1 \ \{p_1\} \quad \text{Assignment} \\
  &7 \quad \{p_1[i+1/i][s+i+1/s]\} s := s + i + 1 \ \{p_1[i+1/i]\} \quad \text{Assignment} \\
  &8 \quad \{p_1[i+1/i][s+i+1/s]\} S_1 \ \{p_1\} \quad \text{Sequence 7, 6} \\
  &9 \quad p_1 \land i < n \rightarrow p_1[i+1/i][s+i+1/s] \quad \text{Predicate logic} \\
  &10 \quad \{p_1 \land i < n\} S_1 \ \{p_1\} \quad \text{Precond. str., 9, 8} \\
  &11 \quad \{\text{inv } p_1\} \text{while } i < n \text{ do } S_1 \text{ od } \ \{p_1 \land i \geq n\} \quad \text{while loop, 10} \\
  &12 \quad \{n \geq 0\} \ i := 0; s := 0; W \ \{p_1 \land i \geq n\} \quad \text{Sequence 5, 11} \\
  &\quad \quad \text{(where } W \text{ is the loop in line 11)} \\
  &13 \quad p_1 \land i \geq n \rightarrow s = \text{sum}(0, n) \quad \text{Predicate logic} \\
  &14 \quad \{n \geq 0\} \ i := 0; s := 0; W \ \{s = \text{sum}(0, n)\} \quad \text{Postcond. weak., 12, 13}
  \end{align*}

- The proof uses two substitutions:
  - $p_1[i+1/i] \equiv 0 \leq i + 1 \leq n \land s = \text{sum}(0, i+1)$
  - $p_1[i+1/i][s+i+1/s] \equiv (0 \leq i \leq n \land s = \text{sum}(0, i+1))[s+i+1/s]$
    $\equiv 0 \leq i + 1 \leq n \land s + i + 1 = \text{sum}(0, i+1)$

- The proof also gives us three “predicate logic obligations” (implications we need to be true, otherwise the overall proof is incorrect). Happily, all three are in fact valid.
- $n \geq 0 \land i = 0 \land s = 0 \rightarrow p_1$
  - I.e., $n \geq 0 \land i = 0 \land s = 0 \rightarrow 0 \leq i \leq n \land \text{sum}(0, i)$
- $p_1 \land i < n \rightarrow p_1[i+1/i][s+i+1/s]$
  - I.e., $(0 \leq i \leq n \land s = \text{sum}(0, i)) \land i < n \rightarrow 0 \leq i+1 \leq n \land s+i+1 = \text{sum}(0, i+1)$
- $p_1 \land i \geq n \rightarrow s = \text{sum}(0, n)$
  - I.e., $(0 \leq i \leq n \land s = \text{sum}(0, i)) \land i \geq n \rightarrow s = \text{sum}(0, n)$
- The order of the lines in the proof is somewhat arbitrary — you can only refer to lines above you in the proof, but they can be anywhere above you.
  - For example, lines 1 and 2 don’t have to be in that order, they just have to be before we use them in the sequence rule at line 3 (which in turn has to be somewhere before line 5, and so on).

G. Full Proof Outlines

- Formal proofs are long and contain repetitive information (we keep copying the same conditions over and over).
- In a proof outline, we add conditions to the inside of the program, not just the ends.
- A full proof outline is a way to write out all the information that you would need to generate a formal proof of correctness.
  - Each correctness triple in the proof must appear in the outline. Said another way, every statement must be part of a triple (including sequences, conditionals, and loops), and every triple must be provable using the proof rules.
  - Precondition strengthening appears as a triple with an extra precondition; postcondition weakening appears as a triple with an extra postcondition.
  - If two conditions sit next to each other, as in $(p_1 \mid p_2)$, it stands for a predicate calculus obligation of $p_1 \rightarrow p_2$.

Example 1: $\{T\} i := 0; \{i = 0\} x := 1 \{i \geq 0 \land x = 2^i\}$ is a full proof outline for the proof

1. $\{T\} i := 0 \{i = 0\}$ Assignment
2. $\{i = 0\} x := 1 \{i = 0 \land x = 1\}$ Assignment
3. $i = 0 \land x = 1 \rightarrow i \geq 0 \land x = 2^i$ Predicate logic
4. $\{i = 0\} x := 1 \{i \geq 0 \land x = 2^i\}$ Postcond. weak., 2, 3
5. $\{T\} i := 0; x := 1 \{i \geq 0 \land x = 2^i\}$ Sequence 1, 4

- A proof outline does not stand for a unique proof. Aside from permuting line orderings, the timing of precondition strengthening and postcondition weakening may not be unique.
- **Example 2**: In Example 1, we could have formed the sequence \(i := 0; x := 1\) and then weakened its postcondition.

   1. \(\{ T \} \ i := 0 \ \{ i = 0 \}\)  
      Assignment
   2. \(\{ i = 0 \} \ x := 1 \ \{ i = 0 \land x = 1 \}\)  
      Assignment
   3. \(\{ T \} \ i := 0 \ ; \ x := 1 \ \{ i = 0 \land x = 1 \}\)  
      Sequence 1, 2
   4. \(i = 0 \land x = 1 \rightarrow i \geq 0 \land x = 2^i\)  
      Predicate logic
   5. \(\{ T \} \ i := 0 \ ; \ x := 1 \ \{ i \geq 0 \land x = 2^i \}\)  
      Postcond. weak. 3, 4

- **Example 3**: Here's a full proof outline for the summation loop; note how the structure of the outline follows the correctness proof.

\[
\{ n \geq 0 \}\ i := 0 ; \ \{ n \geq 0 \land i = 0 \}\ s := 0 ; \ \{ n \geq 0 \land i = 0 \land s = 0 \}\n\]

\[
\{ \text{inv} \ p_1 \equiv 0 \leq i \leq n \land s = \text{sum}(0, i) \} \n\]

\[
\text{while } i < n \ do \ \{ p_1 \land i < n \}\ n \ s := \text{sum}(0, n) \]

\[
\}
\]

- The triples and predicate obligations from this outline are below. They have been listed in the same order as above, in the formal proof of the summation program.

   1. \(\{ n \geq 0 \}\ i := 0 \ \{ n \geq 0 \land i = 0 \}\)
   2. \(\{ n \geq 0 \land i = 0 \}\ s := 0 \ \{ n \geq 0 \land i = 0 \land s = 0 \}\)
   3. \(\{ n \geq 0 \}\ i := 0 ; \ s := 0 \ \{ n \geq 0 \land i = 0 \land s = 0 \}\)
   4. \(n \geq 0 \land i = 0 \land s = 0 \rightarrow p_1\)
   5. \(\{ n \geq 0 \}\ i := 0 ; \ s := 0 \ \{ p_1 \}\)
   6. \(\{ p_1[i+1/i] \}\ i := i+1 \ \{ p_1 \}\)
   7. \(\{ p_1[i+1/i][s+i+1/s] \}\ s := s+i+1 \ \{ p_1[i+1/i] \}\)
   8. \(\{ p_1[i+1/i][s+i+1/s] \}\ s := s+i+1 ; \ i := i+1 \ \{ p_1 \}\)
   9. \(p_1 \land i < n \rightarrow p_1[i+1/i][s+i+1/s] \)
   10. \(\{ p_1 \land i < n \}\ s := s+i+1 ; \ i := i+1 \ \{ p_1 \}\)
   11. \(\{ \text{inv} \ p_1 \}\ W \ \{ p_1 \land i \geq n \}\ where \ W \equiv \text{while } i < n \ do \ s := s+i+1 ; \ i := i+1 \od \)
   12. \(\{ n \geq 0 \}\ i := 0 ; \ s := 0 ; \ \{ \text{inv} \ p_1 \}\ W \ \{ p_1 \land i \geq n \}\)
   13. \(p_1 \land i \geq n \rightarrow s = \text{sum}(0, n) \)
   14. \(\{ n \geq 0 \}\ i := 0 ; \ s := 0 ; \ \{ \text{inv} \ p_1 \}\ W \ \{ s = \text{sum}(0, n) \}\)
H. Minimal Proof Outlines

- If you think about it, you'll realize that most of a full proof outline can be inferred from the structure of the program.
- In a minimal proof outline, we provide the minimum amount of program annotation that allows us to infer the rest of the formal proof outline.
- In general, we can't infer the initial precondition and initial postcondition, nor can we infer the invariants of loops.
- Example 4: Here's the full proof outline from the previous example, with the removable parts in green:

```
{n ≥ 0}
  i := 0; {n ≥ 0 ∧ i = 0}
  s := 0; {n ≥ 0 ∧ i = 0 ∧ s = 0}
  {inv p₁ ≡ 0 ≤ i ≤ n ∧ s = sum(0, i)}
  while i < n do
    {p₁ ∧ i < n} {p₁[i+1/i][s+i+1/s]}
    s := s+i+1; {p₁[i+1/i]}
    i := i+1 {p₁}
  od
  {p₁ ∧ i ≥ n} {s = sum(0, n)}
```

- Dropping the removable parts leaves us with

```
{n ≥ 0} i := 0; s := 0;
  {inv p₁ ≡ 0 ≤ i ≤ n ∧ s = sum(0, i)}
  while i < n do
    s := s+i+1; i := i+1
  od
  {s = sum(0, n)}
```

- In a language like C or Java, the conditions become comments; something like:

```
// Assume: n ≥ 0
int i, s;       // 0 ≤ i ≤ n and s = sum(0, i)
i = s = 0;      // establish i, s
while (i < n) {
  s += i++;    // Get closer to termination
              // and re-establish i, s
}
// Established: s = sum(0, n)
```

- Just as a full proof outline might not stand for a unique proof, a minimal proof outline might not stand for a unique full proof outline.
Example 5: The three full proof outlines

\[ \{ T \} \{ 0 \geq 0 \land i = 2 \} \quad i := 0 \begin{array}{l} \{ T \} \{ 0 \geq 0 \land i = 2 \} \end{array} \quad x := 1 \begin{array}{l} \{ i \geq 0 \land x = 2 \} \end{array} \quad \text{and} \\
\{ T \} \begin{array}{l} i := 0 \end{array} \begin{array}{l} \{ i = 0 \} \quad x := 1 \begin{array}{l} \{ i = 0 \} \end{array} \quad \{ i \geq 0 \land x = 2 \} \end{array} \begin{array}{l} \{ i \geq 0 \land x = 2 \} \end{array} \end{array} \]

both have the same minimal proof outline: \[ \{ T \} \begin{array}{l} i := 0 \end{array} \begin{array}{l} \{ i \geq 0 \land x = 2 \} \end{array} \]

Example 6: The minimal proof outline for

\[ \{ y = x \} \begin{array}{l} \text{if } x < 0 \text{ then } \\ \begin{array}{l} \{ y = x \land x < 0 \} \quad \{ -x = \text{abs}(x) \} \quad y := -x \{ y = \text{abs}(x) \} \\
\quad \text{else } \\ \begin{array}{l} \{ y = x \land x \geq 0 \} \quad \{ y = \text{abs}(x) \} \quad \text{skip} \{ y = \text{abs}(x) \} \\
\quad \text{fi} \\
\quad \{ y = \text{abs}(x) \} \end{array} \\
\quad \text{is } \{ y = x \} \begin{array}{l} \text{if } x < 0 \text{ then } y := -x \text{ fi } \{ y = \text{abs}(x) \} \end{array} \]

Example 7: The minimal proof outline for

\[ \{ n \geq 0 \} \begin{array}{l} j := n; \{ n \geq 0 \land j = n \} \quad s := n; \{ n \geq 0 \land j = n \land s = n \} \\
\quad \{ \text{inv } p \equiv 0 \leq j \leq n \land s = \text{sum}(j, n) \} \\
\quad \text{while } j > 0 \text{ do } \\
\quad \quad \{ p \land j > 0 \} \begin{array}{l} \{ p \} \quad v := e \quad \{ q \} \end{array} \\
\quad \quad \quad \begin{array}{l} s := s + j \{ p \} \\
\quad \quad \quad \{ p \land j \leq 0 \} \quad s = \text{sum}(0, n) \end{array} \end{array} \]

is \[ \{ n \geq 0 \} \begin{array}{l} j := n; \{ n \geq 0 \land j = n \} \quad s := n; \begin{array}{l} \{ n \geq 0 \land j \leq n \land s = \text{sum}(j, n) \} \\
\quad \{ \text{inv } p \equiv 0 \leq j \leq n \land s = \text{sum}(j, n) \} \\
\quad \text{while } j > 0 \text{ do } \\
\quad \quad \{ j := j - 1; \quad \{ p \} \quad \text{skip} \{ p \} \\
\quad \quad \quad \{ s := s + j \} \\
\quad \quad \quad \{ s = \text{sum}(0, n) \} \end{array} \end{array} \]

I. Expanding Partial Proof Outlines

To expand a partial proof outline into a full proof outline, basically we need to infer all the missing conditions. Postconditions are inferred from preconditions using \( sp(\ldots) \), and preconditions are inferred from postconditions using \( wp(\ldots) \). Loop invariants tell us how to annotate the loop body and postcondition, and the test for a conditional statement can become part of a precondition.

A deterministic algorithm isn't possible because a partial proof outline can stand for different proof outlines.

For example, \( \{ p \} v := e \{ q \} \) can become

- \( \{ p \} \{ wp(v := e, q) \} v := e \{ q \} \) or
- \( \{ p \} v := e \{ sp(p, v := e) \} \{ q \} \)
With that warning, here’s an informal algorithm:

**Until every statement can be proved by a triple, apply one of the cases below:**

**Add a precondition:**
1. Prepend \( wp(v := e, q) \) to \( v := e \{ q \} \).
2. Prepend \( q \) to \( \text{skip} \{ q \} \).
3. Prepend some \( p \) to \( S_2 \) in \( S_1 \); \( S_2 \{ q \} \) to get \( S_1 \); \( \{ p \} S_2 \{ q \} \).
4. Add preconditions to the branches of an \textbf{if-else}:
   - Turn \( \{ p \} \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \) into \( \{ p \} \textbf{if } B \textbf{ then } \{ p \wedge B \} S_1 \textbf{ else } \{ p \wedge \neg B \} S_2 \textbf{ fi} \)
5. Add a precondition to an \textbf{if-else}:
   - Prepend \(( B \rightarrow p_1 \land \neg B \rightarrow p_2 )\) to \( \textbf{if } B \textbf{ then } \{ p_1 \} S_1 \textbf{ else } \{ p_2 \} S_2 \textbf{ fi} \)

**Add a postcondition:**
6. Append \( sp(p, v := e) \) to \( \{ p \} v := e \).
7. Append \( p \) to \( \{ p \} \text{skip} \).
8. Append some \( q \) to \( S_1 \) in \( \{ p \} S_1 \); \( S_2 \) to get \( \{ p \} S_1 \{ q \} S_2 \).
9. Add postconditions to the branches of a conditional statement:
   - Turn \( \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \{ q \} \) into \( \textbf{if } B \textbf{ then } S_1 \{ q \} \textbf{ else } S_2 \{ q \} \textbf{ fi} \{ q \} \)
   - Or turn \( \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \{ q_1 \lor q_2 \} \) into
     \( \textbf{if } B \textbf{ then } S_1 \{ q_1 \} \textbf{ else } S_2 \{ q_2 \} \textbf{ fi} \{ q_1 \lor q_2 \} \)
10. Add a postcondition to a conditional statement
    - Append \( q_1 \lor q_2 \) to \( \textbf{if } B \textbf{ then } S_1 \{ q_1 \} \textbf{ else } S_2 \{ q_2 \} \textbf{ fi} \)

**Add loop conditions:**
11. Take a loop and add pre-post-conditions to the loop body and a postcondition to the whole loop:
    - Turn \( \textbf{inv } p \textbf{ while } B \textbf{ do } S_1 \textbf{ od} \) into \( \textbf{inv } p \textbf{ while } B \textbf{ do } \{ p \wedge B \} S_1 \{ p \} \textbf{ od } \{ p \wedge \neg B \} \)

**Weaken or strengthen some condition:**
12. Turn \( \ldots \{ p \} \ldots \) into \( \ldots \{ q \} \ldots \) for some predicate \( q \) where \( p \rightarrow q \).
13. Turn \( \ldots \{ p \} \ldots \) into \( \ldots \{ q \} \ldots \) for some predicate \( p \) where \( p \rightarrow q \).

// End loop

- Using the rules above, a new precondition gets added to the right of the old precondition; a new postcondition gets added to the left of the old postcondition:
  - E.g., taking the \( wp \) of the assignment \( \{ p \} v := e \{ q \} \) gives us \( \{ p \} \{ wp(v := e, q) \} v := e \{ q \} \), not \( \{ wp(v := e, q) \} \{ p \} v := e \{ q \} \).
Example 7 reversed:

- Let's expand

\[ \{ n \geq 0 \} j := n; \ s := n; \]
\[ \{ \text{inv } p \equiv 0 \ \leq j \leq n \ \land \ s = \text{sum}(j, n) \} \]

\begin{verbatim}
while j > 0 do
  j := j-1;
  s := s+j
od
\end{verbatim}
\[ \{ s = \text{sum}(0, n) \} \]

- First, we can apply case 6 (sp of an assignment) to \( j := n \) and to \( s := n \) to get

\[ \{ n \geq 0 \} j := n; \ (n \geq 0 \ \land \ j = n) \ s := n; \ (n \geq 0 \ \land \ j = n \ \land \ s = n) \]
\[ \{ \text{inv } p \equiv 0 \ \leq j \leq n \ \land \ s = \text{sum}(j, n) \} \]

\begin{verbatim}
while j > 0 do
  j := j-1;
  s := s+j
od
\end{verbatim}
\[ \{ s = \text{sum}(0, n) \} \]

- The next three steps are independent of the first two steps we took: First, apply case 11 to the loop:

\[ \{ n \geq 0 \} j := n; \ (n \geq 0 \ \land \ j = n) \ s := n; \ (n \geq 0 \ \land \ j = n \ \land \ s = n) \]
\[ \{ \text{inv } p \equiv 0 \ \leq j \leq n \ \land \ s = \text{sum}(j, n) \} \]

\begin{verbatim}
while j > 0 do
  \{ p \land \ j > 0 \}
  j := j-1;
  s := s+j \{ p \}
od
\end{verbatim}
\[ \{ p \land \ j \leq 0 \} \{ s = \text{sum}(0, n) \} \]

- Then apply case 1 (wp of an assignment) to \( s := s+j \) and to \( j := j-1 \):

\[ \{ n \geq 0 \} j := n; \ (n \geq 0 \ \land \ j = n) \ s := n; \ (n \geq 0 \ \land \ j = n \ \land \ s = n) \]
\[ \{ \text{inv } p \equiv 0 \ \leq j \leq n \ \land \ s = \text{sum}(j, n) \} \]

\begin{verbatim}
while j > 0 do
  \{ p \land \ j > 0 \} \{ p[s+j/s][j-1/j] \}
  j := j-1; \ {p[s+j/s]}
  s := s+j \{ p \}
od
\end{verbatim}
\[ \{ p \land \ j \leq 0 \} \{ s = \text{sum}(0, n) \} \]

- And this finishes the expansion.
Other Features of Expansion

- When we have a sequence of assignments, we can get a number of different proof outlines. Which one to use is pretty much a style issue.

- **Example 8**: Example 7 reversed, we took
  \[
  \{ n \geq 0 \} \ j := n; \ s := n \ (p \equiv 0 \leq j \leq n \wedge s = \text{sum}(j, n))
  \]
  and applied case 6 \((sp)\) to both assignments to get
  \[
  \{ n \geq 0 \} \ j := n; \ {n \geq 0 \wedge j = n} \ s := n; \ {n \geq 0 \wedge j = n \wedge s = n}(p)
  \]
  - Another possibility would have been to use case 1 \((wp)\) on both assignments; we would have gotten
  \[
  \{ n \geq 0 \} \ j := n; \ {0 \leq n \leq n \wedge n = \text{sum}(n, n)} \ s := n \ {0 \leq j \leq n \wedge s = \text{sum}(j, n)}
  \]
  - Or we could have used case 6 \((sp)\) on the first assignment and case 1 \((wp)\) on the second:
  \[
  \{ n \geq 0 \} \ j := n; \ {n \geq 0 \wedge j = n} \ {0 \leq j \leq n \wedge n = \text{sum}(j, n)} \ s := n \ {p}
  \]
  - The three versions produce slightly different predicate logic obligations, but they're all about equally easy to prove.
  - All three versions have essentially the same predicate logic obligations; they just have different syntactic forms:
    - \((sp)\) and \((sp)\): \(n \geq 0 \wedge j = n \wedge s = n \rightarrow 0 \leq j \leq n \wedge s = \text{sum}(j, n)\)
    - \((wp)\) and \((wp)\): \(n \geq 0 \rightarrow 0 \leq n \vee n = \text{sum}(n, n)\)
    - \((sp)\) and \((wp)\): \(n \geq 0 \wedge j = n \rightarrow 0 \leq j \leq n \wedge n = \text{sum}(j, n)\)
  - Similarly, with conditionals \(p\) if \(B\) then \(p_1\) else \(p_2\) \(S_1\) fi can become
    - \(p\) if \(B\) then \(p \wedge B\) \(p_1\) else \(p \wedge \neg B\) \(p_2\) fi via case 4 or
    - \(p\) if \((B \rightarrow p_1) \wedge (\neg B \rightarrow p_2)\) then \(p_1\) else \(p_2\) fi via case 5.
  - We get different predicate logic obligations for the two approaches
    - For the first one, we need \(p \wedge B \rightarrow p_1\) and \(p \wedge \neg B \rightarrow p_2\)
    - For the second one, we need \(p \rightarrow (B \rightarrow p_1) \wedge (\neg B \rightarrow p_2)\)
    - But the work involved in proving the single second condition is about as hard as the combined work of proving the two first conditions.
A. Why
- A formal proof lets us write out in detail the reasons for believing that something is valid.

B. Objectives
At the end of this activity assignment you should be able to
- Write and check formal proofs of correctness.

C. Problems
1. The formal proof below is incomplete; fill in the missing rule names (and line references, where needed).
   1. \( T \rightarrow 0 \geq 0 \land 1 = 2^0 \) __________
   2. \( \{0 \geq 0 \land 1 = 2^0\} \ i := 0 \{i \geq 0 \land 1 = 2^i\} \) __________
   3. \( \{T\} \ i := 0 \{i \geq 0 \land 1 = 2^i\} \) __________
   4. \( \{i \geq 0 \land 1 = 2^i\} \ x := 1 \{i \geq 0 \land x = 2^i\} \) __________
   5. \( \{T\} \ i := 0; \ x := 1 \{i \geq 0 \land x = 2^i\} \) __________

   Here’s an alternate version of the proof that uses forward assignments:
   1. \( \{T\} \ i := 0 \{i = 0\} \) __________
   2. \( \{i = 0\} \ x := 1 \{i = 0 \land x = 1\} \) __________
   3. \( \{T\} \ i := 0; \ x := 1 \{i = 0 \land x = 1\} \) __________
   4. \( i = 0 \land x = 1 \rightarrow i \geq 0 \land x = 2^i \) __________
   5. \( \{T\} \ i := 0; \ x := 1 \{i \geq 0 \land x = 2^i\} \) __________

2. Repeat Problem 1 on the incomplete proof below.
   1. \( \{-x = \text{abs}(x)\} \ y := -x \{y = \text{abs}(x)\} \) __________
   2. \( y = x \land x < 0 \rightarrow -x = \text{abs}(x) \) __________
   3. \( \{y = x \land x < 0\} \ y := -x \{y = \text{abs}(x)\} \) __________
   4. \( \{y = \text{abs}(x)\} \text{skip} \{y = \text{abs}(x)\} \) __________
   5. \( y = x \land x \geq 0 \rightarrow y = \text{abs}(x) \) __________
   6. \( \{y = x \land x \geq 0\} \text{skip} \{y = \text{abs}(x)\} \) __________
   7. \( \{y = x\} \text{if} \ x < 0 \text{then} \ y := -x \\text{fi} \{y = \text{abs}(x)\} \) __________
3. Repeat Problem 1 on the incomplete proof below.

   Below, let \( W \equiv \text{while} \ j > 0 \ do \ j := j - 1; \ s := s + j \ od \)
   
   1. \( \{n \geq 0\} \ j := n \ \{n \geq 0 \land j = n\} \)
   
   2. \( \{n \geq 0 \land j = n\} \ s := n \ \{n \geq 0 \land j = n \land s = n\} \)
   
   3. \( \{n \geq 0\} \ j := n; \ s := n \ \{n \geq 0 \land j = n \land s = n\} \)
   
   4. \( n \geq 0 \land j = n \land s = n \rightarrow p \)
   
   5. \( \{n \geq 0\} \ j := n; \ s := n \ \{p\} \)
   
   6. \( \{p[s + j/s]\} s := s + j \ \{p\} \)
   
   7. \( \{p[s + j/s][j-1/j]\} j := j - 1 \ \{p[s + j/s]\} \)
   
   8. \( p \land j > 0 \rightarrow p[s + j/s][j-1/j] \)
   
   9. \( \{p \land j > 0\} j := j - 1; \ s := s + j \ \{p\} \)
   
   10. \( \{\text{inv} \ p\} W \ \{p \land j \leq 0\} \)
   
   11. \( p \land j \leq 0 \rightarrow s = \text{sum}(0, n) \)
   
   12. \( \{\text{inv} \ p\} W \ \{s = \text{sum}(0, n)\} \)
   
   13. \( \{n \geq 0\} j := n; \ s := n; \ \{\text{inv} \ p\} W \ \{s = \text{sum}(0, n)\} \)

4. Write a formal proof of correctness for \( \{n > 1\} x := n; \ x := x \times x \ \{x \geq 4\} \) that uses wp and precondition strengthening.

5. Repeat Problem 4 but use \( sp \) and postcondition weakening.

For Problems 7–9, you are given a full proof outline; write a corresponding proof of correctness from it. There are multiple right answers.

7. \( \{T\} \ \{0 \geq 0 \land 1 = 2^0\} i := 0; \ \{i \geq 0 \land 1 = 2^i\} x := 1 \ \{i \geq 0 \land x = 2^i\} \)

8a. \( \{y = x\} \text{if} \ x < 0 \text{then} \)
   \( \{y = x \land x < 0\} \{-x = \text{abs}(x)\} y := -x \ \{y = \text{abs}(x)\} \)
   \( \text{else} \)
   \( \{y = x \land x \geq 0\} \{y = \text{abs}(x)\} \text{skip} \ \{y = \text{abs}(x)\} \)
   \( \text{fi} \ \{y = \text{abs}(x)\} \)

8b. \( \{y = x\} \text{if} \ x < 0 \text{then} \)
   \( \{y = x \land x < 0\} y := -x \ \{y_0 = x \land x < 0 \land y = -x\} \)
   \( \text{else} \)
   \( \{y = x \land x \geq 0\} \text{skip} \ \{y = x \land x \geq 0\} \)
   \( \text{fi} \ \{(y_0 = x \land x < 0 \land y = -x) \lor (y = x \land x \geq 0)\} \{y = \text{abs}(x)\} \)

CS 536: Science of Programming – 12 – © James Sasaki, 2018
8c. \( \{ y = x \} \) \((x < 0 \rightarrow -x = \text{abs}(x)) \wedge (x \geq 0 \rightarrow y = \text{abs}(x))\) 

\[ \text{if } x < 0 \text{ then} \]
\[ \{ -x = \text{abs}(x) \} \quad y := -x \quad \{ y = \text{abs}(x) \} \]

\[ \text{else} \]
\[ \{ y = \text{abs}(x) \} \quad \textbf{skip} \quad \{ y = \text{abs}(x) \} \]

\[ \text{fi} \quad \{ y = \text{abs}(x) \} \]

9. Hint: Use \( sp \) for the two loop initialization assignments.

\[ \{ n \geq 0 \} \quad j := n; \quad \{ n \geq 0 \wedge j = n \} \quad s := n; \quad \{ n \geq 0 \wedge j = n \wedge s = n \} \]

\[ \{ \text{inv } p \equiv 0 \leq j \leq n \wedge s = \text{sum}(j,n) \} \]

\[ \text{while } j > 0 \text{ do} \]
\[ \{ p \wedge j > 0 \} \quad \{ p \text{[s+j]/s}[j-1/j] \} \quad j := j-1; \]
\[ \{ p \text{[s+j]/s] s := s+j \} \quad \{ p \} \]

\[ \text{od} \quad \{ p \wedge j \leq 0 \} \quad \{ s = \text{sum}(0,n) \} \]

For Problems 10 – 12, you are given a minimal proof outline and should expand it to a full proof outline. Don't give the formal proof of correctness. There may be multiple right answers; any right answer is sufficient.

10. \( \{ n > 1 \} \quad i := 1; \quad s := 0 \quad \{ 0 \leq i < n \wedge s = \text{sum}(0,i-1) \} \)

11. \( \{ T \} \quad \textbf{if } x \geq 0 \text{ then } y := x \quad \textbf{else } y := -x \quad \text{fi} \quad \{ y = \text{abs}(x) \} \)

12. The program below has a bug. In addition to expanding its minimal proof outline, answer the following question: Where in the full proof outline does the bug appear? (I.e., some part of the formal proof would fail; where in the full proof outline do we see what that part would be?) Give a way to fix the bug.

\[ \{ \text{inv } p \equiv 0 \leq i \leq n+1 \wedge s = \text{sum}(0,i-1) \} \]

\[ \text{while } i \leq n \text{ do} \]
\[ i := i+1; \]
\[ s := s+i \]

\[ \text{od} \quad \{ s = \text{sum}(0,n) \} \]