

Proofs and Proof Outlines for Partial Correctness

Part 2: Partial and Minimal Proof Outlines

CS 536: Science of Programming, Spring 2021

3/23 p.2

A. Why

- A formal proof lets us write out in detail the reasons for believing that something is valid.
- Proof outlines condense the same information as a proof.

B. Objectives

At the end of this class you should

- Know the structure of full proof outlines and formal proofs and how they are related.
- Know the difference between full, partial, and minimal proof outlines and how they are related.

C. Minimal Proof Outlines

- In a full proof outline of correctness, we include all the triples found in a formal proof of correctness, but we omit much of the redundant text, which makes them much easier to work with than formal proofs. But if you think about it, you'll realize that we can shorten the outline by omitting conditions that can be inferred to exist 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, so a minimal outline will include those conditions and possibly no others.
- A partial proof outline is somewhere in the middle: More filled-in than a minimal outline but not completely full.

Example 1

- Here's the full proof outline from the previous class, with the removable parts in green:

\[
\begin{align*}
\{n \geq 0\} \\
\{n \geq 0 \land k = 0\} & \quad \text{// Inferred as the sp of } k := 0 \\
\{n \geq 0 \land k = 0 \land s = 0\} & \quad \text{// Inferred as the sp of } s := 0 \\
\{\text{inv } p_1 \equiv 0 \leq k \leq n \land s = \text{sum}(0, k)\} & \quad \text{// Can't be inferred} \\
\text{while } k < n \text{ do} & \\
\{p_1 \land k < n\} & \quad \text{// Loop rule requires } \text{inv} \land \text{ loop test at top of loop body} \\
\{p_1[k + 1/k][s+k+1/s]\} & \quad \text{// Inferred as the wp of } s := s+k+1 \\
\{s := s+k+1; \ p_1[k+1/k]\} & \quad \text{// Inferred as the wp of } k := k+1 \\
\{k := k+1 \ \{p_1\}\} & \quad \text{// Loop rule requires invariant at bottom of loop body} \\
\od & \\
\{p_1 \land k \geq n\} & \quad \text{// Loop rule requires } \text{inv} \land \neg \text{ loop test after the loop}
\end{align*}
\]
• Dropping the inferable parts leaves us with the minimal outline:

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

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

```c
// Assume: n \geq 0
int k, s;  // 0 \leq k \leq n and s = \text{sum}(0,k)
k = s = 0;  // establish k, s
while (k < n) {
    s += k+1;  // reset s [3/23]
    ++k;      // Get closer to termination and reestablish k, s
}
// Established: s = \text{sum}(0, n)
```

• The following example shows how different total proof outlines can all have the same minimal proof outline.

**Example 2**

• The three full proof outlines

\[
\begin{align*}
\{T\} & \quad \{0 \geq 0 \land 1 = 2^0\} \quad k := 0; \quad \{k \geq 0 \land 1 = 2^k\} \quad x := 1 \quad \{k \geq 0 \land x = 2^k\} \\
\{T\} & \quad k := 0; \quad \{k = 0\} \quad x := 1 \quad \{k = 0 \land x = 1\} \quad \{k \geq 0 \land x = 2^k\} \\
\{T\} & \quad k := 0; \quad \{k = 0\} \quad \{k \geq 0 \land 1 = 2^k\} \quad x := 1 \quad \{k \geq 0 \land x = 2^k\}
\end{align*}
\]

all have the same minimal proof outline, \(\{T\} \quad k := 0; \quad x := 1 \quad \{k \geq 0 \land x = 2^k\}\)

• The reason multiple full proof outlines can have the same minimal outline is because different organizations of \(wp\) and \(sp\) can have the same minimal outline. There can also be differences in whether and where preconditions are strengthened or postconditions are weakened.

**Example 3**

• The minimal proof outline for

\[
\{y = x\}
\]

\[\text{if } x < 0 \text{ then}\]

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

\[\text{else}\]

\[
\{y = x \land x \geq 0\} \quad \text{skip} \quad \{y = x \land x \geq 0\} \quad \{y = \text{abs}(x)\}
\]

\[\text{fi}\]

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

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

• Note this is the same minimal outline for the following full outline for the same code:
\{y = x\}
\{(x < 0 \rightarrow -x = \text{abs}(x)) \land (x \geq 0 \rightarrow y = \text{abs}(x))\} \quad // \text{wp of the if-else}
if x < 0 then
\{-x = \text{abs}(x)\} y := -x \{y = \text{abs}(x)\}
else
\{y = \text{abs}(x)\} \textbf{skip} \{y = \text{abs}(x)\}
fi
\{y = \text{abs}(x)\}

Example 4
• The minimal proof outline for
\{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 = 0 \leq j \leq n \land s = \text{sum}(j, n)\}
while j > 0 do
\{p \land j > 0\} \{p[s+j/s][j–1/j]\}
\quad j := j–1; \{p[s+j/s]\}
\quad s := s+j \{p\}
\od
\{p \land j \leq 0\} \{s = \text{sum}(0, n)\}

is
\{n \geq 0\} j := n; s := n;
\{\text{inv} p = 0 \leq j \leq n \land s = \text{sum}(j, n)\}
while j > 0 do
\quad j := j–1; s := s+j
\od
\{s = \text{sum}(0, n)\}

D. 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 \textit{sp}(\ldots), and preconditions are inferred from postconditions using \textit{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.
• Expanding a partial outline can lead to a number of different full outlines, but all the full outlines will be correct, and the differences between them are generally stylistic. Expansion can have different results because multiple full outlines can have the same minimal outline.
• For example, since \{p\} \{\text{wp}(v := e, q)\} v := e \{q\} and \{p\} v := e \{\text{sp}(p, v := e)\} \{q\} have the same minimal outline, \{p\} v := e \{q\} can expand to either full outline.
• The situation similar to how a full proof outline can expand to various (but all correct) formal proofs, but the different proofs simply shuffle the order of the rule applications. The different full outlines here are actually different, though generally only in small ways.

• So we can't have a deterministic algorithm for expanding minimal outlines, but with that warning, here's an informal nondeterministic algorithm. (Added conditions are shown in green.)

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

**A. 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}-\textbf{else}:
   
   Turn \( \{p\} \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi} \) into \( \{p\} \text{if } B \text{ then } \{p \land B\} S_1 \text{ else } \{p \land \neg B\} S_2 \text{ fi} \)
5. Add a precondition to an \textbf{if}-\textbf{else}:
   
   Prepend \( (B \rightarrow p_1) \land (\neg B \rightarrow p_2) \) to \( \text{if } B \text{ then } \{p_1\} S_1 \text{ else } \{p_2\} S_2 \text{ fi} \)

**B. Or 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 \( \{p\} S_1; S_2 \{q\} \) to get \( \{p\} S_1; \{q\} S_2 \).
9. Add a postcondition to a conditional statement
   
   Append \( q_1 \lor q_2 \) to \( \text{if } B \text{ then } S_1 \{q_1\} \text{ else } S_2 \{q_2\} \text{ fi} \)
10. Add postconditions to the branches of a conditional statement:
    
    Turn \( \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q_1 \lor q_2\} \) into \( \text{if } B \text{ then } S_1 \{q_1\} \text{ else } S_2 \{q_2\} \text{ fi } \{q_1 \lor q_2\} \)

**C. Or add loop conditions:**

11. Take a loop and add pre-and post-conditions to the loop body; add a postcondition for the loop:
    
    Turn \( \{\text{inv } p\} \text{while } B \text{ do } S_1 \text{ od} \) into \( \{\text{inv } p\} \text{while } B \text{ do } \{p \land B\} S_1 \{p\} \text{ od } \{p \land \neg B\} \)

**D. Or strengthen or weaken some condition:**

12. Turn \( \{q\} \text{ ... into } \{p\} \{q\} \text{ ... for some predicate } p \text{ where } p \rightarrow q \).
13. Turn \( \{p\} \text{ ... into } \{p\} \{q\} \text{ ... for some predicate } q \text{ where } p \rightarrow q \).

// End loop

• Using the rules above, any newly added precondition gets added to the right of the old precondition; any newly added 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 4 reversed:**

• Let's expand
\{n \geq 0\} \ j := n; \ s := n;
\{inv \ p = 0 \leq j \leq n \wedge s = \text{sum}(j, \ n)\}
while \ j > 0 \ do
\ j := j-1;
\ s := s+j
od
\{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 \wedge j = n} \ s := n; \ {n \geq 0 \wedge j = n \wedge s = n}\)
\{inv \ p = 0 \leq j \leq n \wedge s = \text{sum}(j, \ n)\}
while \ j > 0 \ do
\ j := j-1;
\ s := s+j
od
\{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 \wedge j = n} \ s := n; \ {n \geq 0 \wedge j = n \wedge s = n}\)
\{inv \ p = 0 \leq j \leq n \wedge s = \text{sum}(j, \ n)\}
while \ j > 0 \ do
\ \{p \wedge j > 0\}
\ j := j-1;
\ s := s+j \ \{p\}
od
\{p \wedge 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\{n \geq 0\} \ j := n; \ {n \geq 0 \wedge j = n} \ s := n; \ {n \geq 0 \wedge j = n \wedge s = n}\)
\{inv \ p = 0 \leq j \leq n \wedge s = \text{sum}(j, \ n)\}
while \ j > 0 \ do
\ \{p \wedge j > 0\} \{p[s+j/s][j-1/j]\}
\ j := j-1; \ \{p[s+j/s]\}
\ s := s+j \ \{p\}
od
\{p \wedge j \leq 0\} \{s = \text{sum}(0, \ n)\}

• And this finishes the expansion.
Other Features of Expansion

- In Example 2, we saw that a number of full proof outlines can have the same minimal proof outline. The inverse is that a partial proof outline might expand into a number of different full proof outlines. Which one to use is pretty much a style issue.

Example 5

- In Example 4 reversed, we took

\[
\begin{align*}
\{n \geq 0\} j &:= n; \ s := n \ \{p = 0 \leq j \leq n \wedge s = sum(j, n)\} \\
\text{and applied case 6 (sp) to both assignments to get} \\
\{n \geq 0\} j &:= n; \ \{n \geq 0 \wedge j = n\} s := n; \ \{n \geq 0 \wedge j = n \wedge s = n\} \ \{p\}
\end{align*}
\]

- Another possibility would have been to use case 1 (wp) on both assignments; we would have gotten

\[
\begin{align*}
\{n \geq 0\} j &:= n; \ \{n \geq 0 \wedge j = n\} s := n; \ \{n \geq 0 \wedge j = n \wedge s = n\} \ \{p\}
\end{align*}
\]

- Or we could have used case 6 (sp) on the first assignment and case 1 (wp) on the second:

\[
\begin{align*}
\{n \geq 0\} j &:= n; \ \{n \geq 0 \wedge j = n\} \ \{0 \leq j \leq n \wedge n = sum(j, n)\} s := n \ \{p\}
\end{align*}
\]

- The three versions produce slightly different predicate logic obligations, but they're all about equally easy to prove.

  - **sp and sp:** \( n \geq 0 \wedge j = n \wedge s = n \rightarrow 0 \leq j \leq n \wedge s = sum(j, n) \)
  - **wp and wp:** \( n \geq 0 \rightarrow 0 \leq n \leq n \wedge n = sum(n, n) \)
  - **sp and wp:** \( n \geq 0 \wedge j = n \rightarrow 0 \leq j \leq n \wedge n = sum(j, n) \)

- Similarly, with a conditional triple \( \{p\} \text{if } B \text{ then } \{p_1\} S_1 \text{ else } \{p_2\} S_2 \text{ fi} \), we can get

  - With case 4: \( \{p\} \text{if } B \text{ then } \{p \wedge B\} \ {p_1}\ S_1 \text{ else } \{p \wedge \neg B\} \ {p_2}\ S_2 \text{ fi} \)
  - Or with case 5: \( \{p\} \ {\{B \rightarrow p_1\} \wedge \{\neg B \rightarrow p_2\}} \text{if } B \text{ then } \{p_1\} S_1 \text{ else } \{p_2\} S_2 \text{ fi} \)

- We get different predicate logic obligations for the two approaches:
  - With case 4: \( p \wedge B \rightarrow p_1 \) and \( p \wedge \neg B \rightarrow p_2 \)
  - With case 5: \( p \rightarrow (B \rightarrow p_1) \wedge (\neg B \rightarrow p_2) \)

- But the work involved in proving the single second condition is about as hard as the combined work of proving the two first conditions.
Proofs and Proof Outlines for Partial Correctness

Part 2: Partial and Minimal Proof Outlines

CS 536: Science of Programming, Spring 2021

A. Why

- 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 activity assignment you should be able to

- Write and check proof outlines of partial correctness.

C. Problems

For Problems 1 - 3, you are given a minimal proof outline and should expand it to a full proof outline. Don't give the formal proof of partial correctness. Do list any predicate logic obligations.

1. \( \{ n > 1 \} \ k := 1; \ s := 0 \ \{ 0 \leq k < n \land s = \text{sum}(0, k-1) \} \)
   a. Use \( \text{wp} \) on both assignments.
   b. Use \( \text{sp} \) on both assignments.
   c. Use \( \text{sp} \) on the left assignment and \( \text{wp} \) on the right assignment.

2. \( \{ T \} \ \text{if } x \geq 0 \ \text{then } y := x \ \text{else } y := -x \ \text{fi } \{ y = \text{abs}(x) \} \)
   a. Use \( \text{sp} \) on both branches and on the \( \text{if-fi} \) as a whole.
   b. Use \( \text{wp} \) on the \( \text{if-fi} \) as a whole and on both branches.
   c. Use \( (P \land B) \) and \( (P \land \neg B) \) (from the conditional rule) as overall preconditions for the two branches, and use \( \text{wp} \) on both branches.

3. Since the invariant of the loop below is a predicate function call, substitutions using it are easy.

\[
\begin{align*}
\{ n \geq 0 \} & \ x := 0; \ y := 1; \\
\{ \text{inv } P(a, b, x, y) \} & \\
\text{while } x < n & \text{ do } \\
& \ x := f(x, y); \\
& \ y := f(y, x) \\
\text{od } \{ a + x < b - y \}
\end{align*}
\]
   a. Use \( \text{wp} \) as much as you can.
   b. Use \( \text{sp} \) as much as you can.
4. Expand the minimal proof outline below. The program has a bug; in the full proof outline, in what line(s) and in what form does the bug appear? Also, give two ways to fix the bug.

\[ \text{inv } p = 0 \leq k \leq n+1 \land s = \text{sum}(0, k-1) \]
\[ \text{while } k \leq n \text{ do} \]
\[ \quad k := k+1; \]
\[ \quad s := s + k \]
\[ \text{od} \]
\[ \{ s = \text{sum}(0, n) \} \]

```
Solution to Practice 17 (Full and Minimal Proof Outlines for Partial Correctness)

1. (Expansions of minimal outline for two assignments.) Note that the three predicate logic obligations below differ, but not in significant ways.

1a. An expansion using \( wp \) (on both assignments):

\[
\{ n > 1 \}
\{ 0 \leq 1 < n \land 0 = \text{sum}(0, 1-1) \} k := 1;
\{ 0 \leq k < n \land 0 = \text{sum}(0, k-1) \} s := 0 \{ 0 \leq k < n \land s = \text{sum}(0, k-1) \}
\]

The predicate logic obligation is \( n > 1 \rightarrow (0 \leq 1 < n \land 0 = \text{sum}(0, 1-1)) \)

1b. An expansion using \( sp \) (on both assignments):

\[
\{ n > 1 \}
\{ 0 \leq k < n \land s = \text{sum}(0, k-1) \}
\]

The predicate logic obligation is \( (n > 1 \land k = 1 \land s = 0) \rightarrow (0 \leq k < n \land s = \text{sum}(0, k-1)) \)

1c. An expansion using \( sp \) (on the left) and \( wp \) (on the right):

\[
\{ n > 1 \}
\{ 0 \leq k < n \land s = \text{sum}(0, k-1) \}
\]

The predicate logic obligation is \( (n > 1 \land k = 1) \rightarrow (0 \leq k < n \land s = \text{sum}(0, k-1)) \)

2. (Expansions of minimal outline for an \( \text{if-else} \).) Note "\( T \land " has been dropped in various places.

2a. An expansion using \( sp \) on both branches and on the whole \( \text{if-else} \):

\[
\{ T \}
\{ x \geq 0 \land y = x \}
\{ x < 0 \land y = -x \}
\]

The predicate logic obligation is \( (x \geq 0 \land y = x) \lor (x < 0 \land y = -x) \rightarrow y = \text{abs}(x) \).

* The decision of where to place line breaks in the full outline is stylistic. Your style can certainly differ. But you should use some consistent indentation.
2b. An expansion using \( wp \) on the whole if-else and both branches:

\[
\{T\} \{x \geq 0 \rightarrow x = \text{abs}(x) \wedge (x < 0 \rightarrow -x = \text{abs}(x))\}
\]

\[
\text{if} \ x \geq 0 \ \text{then}
\]

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

\[
\text{else}
\]

\[
\{-x = \text{abs}(x)\} \ y := -x \ \{y = \text{abs}(x)\}
\]

\[
\text{fi}
\]

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

The logic obligation is \( (T \rightarrow ) \ (x \geq 0 \rightarrow x = \text{abs}(x)) \wedge (x < 0 \rightarrow -x = \text{abs}(x)) \)

2c. An expansion using \( (P \wedge B) \) and \( (P \wedge \neg B) \) as preconditions for the branches and also \( wp \) on the branches:

\[
\{T\}
\]

\[
\text{if} \ x \geq 0 \ \text{then}
\]

\[
\{x \geq 0\} \ {x = \text{abs}(x)} \ y := x \ \{y = \text{abs}(x)\}
\]

\[
\text{else}
\]

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

\[
\text{fi}
\]

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

This time there are two logic obligations, namely, the conjuncts from part (b):

\( (x \geq 0 \rightarrow x = \text{abs}(x)) \) and \( (x < 0 \rightarrow -x = \text{abs}(x)) \)

3. Expansion of a loop. For reference, the minimal outline was

\[
\{n \geq 0\} \ x := 0; \ y := 1; \ \{\text{inv} \ P(a, b, x, y)\} \ \text{while} \ x < n \ \text{do} \ x := f(x, y); \ y := f(y, x) \ \text{od} \ \{a + x < b - y\}
\]

3a. An expansion using \( wp \) as much as possible:

\[
\{n \geq 0\}
\]

\[
\{P(a, b, 0, 1) \ x := 0;\}
\]

\[
\{P(a, b, x, 1)\} \ y := 1;
\]

\[
\{\text{inv} \ P(a, b, x, y)\}
\]

\[
\text{while} \ x < n \ \text{do}
\]

\[
\{x < n \wedge P(a, b, x, f(y, x))\}
\]

\[
\{P(a, b, f(x, y), f(y, f(x, y)))\}
\]

\[
x := f(x, y);
\]

\[
\{P(a, b, x, f(y, x))\} \ y := f(y, x)
\]

\[
\{P(a, b, x, y)\}
\]

\[
\text{od}
\]

\[
\{x \geq n \wedge P(a, b, x, y)\}
\]

\[
\{a + x < b - y\}
\]
There are three predicate logic obligations. Basically, they show that loop initialization works, the loop precondition is met, and the loop establishes the desired postcondition.

\[
(n \geq 0 \rightarrow P(a, b, 0, 1))
\]

\[
(x < n \land P(a, b, x, f(y, x))) \rightarrow P(a, b, f(x, y), f(y, f(x, y)))
\]

\[
(x \geq n \land P(a, b, y) \rightarrow a + x < b - y)
\]

3b. An expansion using \( sp \) as much as possible.

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

\[
y := 1; \{ n \geq 0 \land x = 0 \land y = 1 \}
\]

\[
\{ \text{inv} P(a, b, x, y) \}
\]

\[
\text{while } x < n \text{ do}
\]

\[
\{ x < n \land P(a, b, x, f(y, x)) \}
\]

\[
\{ P(a, b, x_0, y_0) \land y = f(y_0, x_0) \land x = f(x_0, y) \}
\]

\[
x := f(x, y);
\]

\[
\{ P(a, b, x, y_0) \land y = f(y_0, x) \}
\]

\[
y := f(y, x)
\]

\[
\{ P(a, b, x, y) \}
\]

\[
\text{od}
\]

\[
\{ x \geq n \land P(a, b, x, y) \}
\]

\[
\{ a + x < b - y \}
\]

There are again three predicate logic obligations. The third one (loop termination establishes the program's postcondition) is exactly the same as in the \( wp \) version because it's set by the loop framework \( \{ \text{inv} \ldots \} \text{ while } \ldots \), not by the loop body.

\[
(n \geq 0 \land x = 0 \land y = 1 \rightarrow P(a, b, x, y))
\]

\[
(x < n \land P(a, b, x, f(y, x))) \rightarrow P(a, b, x_0, y_0) \land y = f(y_0, x_0) \land x = f(x_0, y)
\]

\[
(x \geq n \land P(a, b, y) \rightarrow a + x < b - y)
\]

4. The expansion is straightforward:

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

\[
\text{while } k \leq n \text{ do}
\]

\[
\{ p \land k \leq n \} \{ p[s+k/s][k+1/k] \} k := k+1;
\]

\[
\{ p[s+k/s] \} s := s+k \{ p \}
\]

\[
\text{od}
\]

\[
\{ p \land k > n \} \{ s = \text{sum}(0, n) \}
\]

The program is not correct because of the predicate obligation \( p \land k \leq n \rightarrow p[s+k/s][k+1/k] \) expands to an invalid predicate:

\[
0 \leq k \leq n+1 \land s = \text{sum}(0, k-1) \land k \leq n \rightarrow 0 \leq k+1 \leq n+1 \land s+(k+1) = \text{sum}(0, k+1-1)
\]
The error is that we need $s+(k+1) = \text{sum}(0, k+1-1)$, but this doesn't follow from $s = \text{sum}(0, k-1)$.

The new value of $s$ is off by one:

$$s = \text{sum}(0, k-1)$$
$$\implies s+k = \text{sum}(0, k-1) + k$$
$$\implies s+k = \text{sum}(0, k)$$
$$\implies s+k+1 = \text{sum}(0, k) + 1$$

One fix is to swap $k := k+1$ and $s := s+k$. Our predicate logic obligation becomes

$$p \land k \leq n \rightarrow p[k+1/k][s+k/s]$$
$$= p \land k \leq n \rightarrow (0 \leq k+1 \leq n+1 \land s = \text{sum}(0, k+1-1)) [s+k/s]$$
$$= 0 \leq k \leq n+1 \land s = \text{sum}(0, k-1) \land k \leq n$$
$$\implies (0 \leq k+1 \leq n+1 \land s+k = \text{sum}(0, k+1-1))$$

Another fix is to change $s := s+k$ to $s := s+k-1$. Our obligation becomes

$$p \land k \leq n \rightarrow p[s+k-1/s][k+1/k]$$
$$= p \land k \leq n \rightarrow (0 \leq k \leq n+1 \land s+k-1 = \text{sum}(0, k-1)) [k+1/k]$$
$$= p \land k \leq n \rightarrow (0 \leq k+1 \leq n+1 \land s+(k+1)-1 = \text{sum}(0, k+1-1))$$