Formal Correctness Proofs and Proof Outlines

CS 536: Science of Programming, Fall 2019

10/18

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 partial correctness.
- Recognize a full proof outline and translate it to a formal proof of partial 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 $s = sum(0, n) = 0+1+\ldots+n$ where $n \geq 0$. (If $n < 0$, define $sum(0, n) = 0$.) Note the loop invariant appears explicitly as a comment.

  {n ≥ 0}
  i := 0; s := 0;
  {**inv** $p_1 \equiv 0 \leq i \leq n \wedge s = sum(0, i)$}
  while i < n **do**
        s := s+i+1; i := i+1
  od
  {s = sum(0, n)}

- Informally, to see that this program works, we need
  - $\{n \geq 0\}$ i := 0; s := 0 $\{p_1 \equiv 0 \leq i \leq n \wedge s = sum(0, i)\}$
  - $\{p_1 \wedge i < n\}$ s := s+i+1; i := i+1 $\{p_1\}$
  - $p_1 \wedge i \geq n \rightarrow s = sum(0, n)$
- 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 $s = sum(0, i)$ in our invariant, so we have the loop terminate with $i = n$.
  - If instead we use $s = sum(0, i+1)$ or $s = sum(0, i-1)$ 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 = sum(0, i)

  ```
  {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)}
  ```

- **Example 2**: Using s = sum(0, i+1)

  ```
  {n > 0}
  i := 0; s := 1;
  {inv p₂ ≡ 0 ≤ i < n ∧ s = sum(0, i+1)}
  while i < n-1 do
      s := s+i+2; i := i+1
  od
  {s = sum(0, n)}
  ```

- **Example 3**: Using s = sum(0, i−1)

  ```
  {n ≥ 0}
  i := 1; s := 0;
  {inv p₂ ≡ 1 ≤ i ≤ n+1 ∧ s = sum(0, i—1)}
  while i ≤ n do
      s := s+i; i := i+1
  od
  {s = sum(0, n)}
  ```

E. Formal Proofs of Partial Correctness

- A formal proof of partial 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 partial 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 = length of XY	Assumption
2.	Angle ABC = Angle XYZ	Assumption
3.	Length of BC = length of YZ	Assumption
4.	Triangles ABC, XYZ are congruent	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**):

  $\{n \geq 0\}$
  $i := 0;\ s := 0;$
  $\{\mathbf{inv}\ p_1 \equiv 0 \leq i \leq n \wedge s = sum(0, i)\}$
  $\mathbf{while}\ i < n\ \mathbf{do}$
  $\quad s := s+i+1;\ i := i+1$
  $\mathbf{od}$
  $\{s = sum(0, n)\}$

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

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

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

- $n \ge 0 \land i = 0 \land s = 0 \to p_1$
  - I.e., $n \ge 0 \land i = 0 \land s = 0 \to 0 \le i \le n \land s = sum(0, i)$
- $p_1 \land i < n \to p_1[i+1/i][s+i+1/s]$
  - I.e., $(0 \le i \le n \land s = sum(0, i)) \land i < n \to 0 \le i+1 \le n \land s+i+1 = sum(0, i+1)$
- $p_1 \land i \ge n \to s = sum(0, n)$
  - I.e., $(0 \le i \le n \land s = sum(0, i)) \land i \ge n \to s = sum(0, n)$

- 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).

--- End 10/16

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 partial correctness.
  - Each 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 $\{p_1\}\{p_2\}$, it stands for a predicate calculus obligation of $p_1 \to p_2$.
- 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 1**: Here we form the sequence `i := 0; x := 1` and then weaken 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.	$i = 0 \land x = 1 \to i \ge 0 \land x = 2^i$	Predicate logic
5.	$\{T\}$ `i := 0; x := 1` $\{i \ge 0 \land x = 2^i\}$	Postcond. weak. 3, 4

  - The full proof outline for this is $\{T\}$ `i := 0;` $\{i = 0\}$ `x := 1` $\{i = 0 \land x = 1\}$ $\{i \ge 0 \land x = 2^i\}$

- **Example 2**: This is like Example 1 but uses precondition strengthening instead of postcondition weakening.

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

  - The full proof outline is $\{T\}$ $\{0 \geq 0 \wedge 1 = 2^0\}$ `i := 0;` $\{i \geq 0 \wedge 1 = 2^i\}$ `x := 1` $\{i \geq 0 \wedge x = 2^i\}$

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

$\{n \geq 0\}$ `i := 0;` $\{n \geq 0 \wedge i = 0\}$ `s := 0;` $\{n \geq 0 \wedge i = 0 \wedge s = 0\}$
$\{\mathbf{inv}\ p_1 \equiv 0 \leq i \leq n \wedge s = sum(0, i)\}$
while $i < n$ **do**
  $\{p_1 \wedge i < n\}$ $\{p_1[i+1/i][s+i+1/s]\}$
  `s := s+i+1;` $\{p_1[i+1/i]\}$
  `i := i+1` $\{p_1\}$
od
$\{p_1 \wedge i \geq n\}$
$\{s = sum(0, n)\}$

- 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` $\{n \geq 0 \wedge i = 0\}$
2. $\{n \geq 0 \wedge i = 0\}$ `s := 0` $\{n \geq 0 \wedge i = 0 \wedge s = 0\}$
3. $\{n \geq 0\}$ `i := 0; s := 0` $\{n \geq 0 \wedge i = 0 \wedge s = 0\}$
4. $n \geq 0 \wedge i = 0 \wedge s = 0 \rightarrow p_1$
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. $p_1 \wedge i < n \rightarrow p_1[i+1/i][s+i+1/s]$
10. $\{p_1 \wedge i < n\}$ `s := s+i+1; i := i+1` $\{p_1\}$
11. $\{\mathbf{inv}\ p_1\}$ W $\{p_1 \wedge i \geq n\}$ where $W \equiv$ **while** $i < n$ **do** `s := s+i+1; i := i+1` **od**
12. $\{n \geq 0\}$ `i := 0; s := 0;` $\{\mathbf{inv}\ p_1\}$ W $\{p_1 \wedge i \geq n\}$
13. $p_1 \wedge i \geq n \rightarrow s = sum(0, n)$
14. $\{n \geq 0\}$ `i := 0; s := 0;` $\{\mathbf{inv}\ p_1\}$ W $\{s = sum(0, n)\}$

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:

  $\{n \geq 0\}$
  $i := 0;$ $\{n \geq 0 \land i = 0\}$
  $s := 0;$ $\{n \geq 0 \land i = 0 \land s = 0\}$
  $\{\mathbf{inv}\ p_1 \equiv 0 \leq i \leq n \land s = sum(0, i)\}$
  while $i < n$ **do**
  $\quad\{p_1 \land i < n\}$ $\{p_1[i+1/i][s+i+1/s]\}$
  $\quad s := s+i+1;$ $\{p_1[i+1/i]\}$
  $\quad i := i+1$ $\{p_1\}$
  od
  $\{p_1 \land i \geq n\}$ $\{s = sum(0, n)\}$

- Dropping the removable parts leaves us with

  $\{n \geq 0\}$ $i := 0;$ $s := 0;$
  $\{\mathbf{inv}\ p_1 \equiv 0 \leq i \leq n \land s = sum(0, i)\}$
  while $i < n$ **do**
  $\quad s := s+i+1;$ $i := i+1$
  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) {
   ++i;         // Get closer to termination
   s += i;      // 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

  $\{T\}\ \{0 \ge 0 \wedge 1 = 2^0\}\ \texttt{i := 0;}\ \{i \ge 0 \wedge 1 = 2^i\}\ \texttt{x := 1}\ \{i \ge 0 \wedge x = 2^i\}$ and

  $\{T\}\ \texttt{i := 0;}\ \{i = 0\}\ \texttt{x := 1}\ \{i = 0 \wedge x = 1\}\ \{i \ge 0 \wedge x = 2^i\}$

  $\{T\}\ \texttt{i := 0;}\ \{i = 0\}\ \{i \ge 0 \wedge 1 = 2^i\}\ \texttt{x := 1}\ \{i \ge 0 \wedge x = 2^i\}$

  both have the same minimal proof outline: $\{T\}\ \texttt{i := 0; x := 1}\ \{i \ge 0 \wedge x = 2^i\}$

- **Example 6**: The minimal proof outline for

  $\{y = x\}$
  if $x < 0$ **then**
  $\quad\{y = x \wedge x < 0\}\ \{-x = abs(x)\}\ \texttt{y := -x}\ \{y = abs(x)\}$
  else
  $\quad\{y = x \wedge x \ge 0\}\ \{y = abs(x)\}$ **skip** $\{y = abs(x)\}$
  fi
  $\{y = abs(x)\}$

  is $\{y = x\}$ **if** $x < 0$ **then** $\texttt{y := -x}$ **fi** $\{y = abs(x)\}$

- **Example 7**: The minimal proof outline for

  $\{n \ge 0\}\ \texttt{j := n;}\ \{n \ge 0 \wedge j = n\}\ \texttt{s := n;}\ \{n \ge 0 \wedge j = n \wedge s = n\}$
  $\{$**inv** $p \equiv 0 \le j \le n \wedge s = sum(j, n)\}$
  while $j > 0$ **do**
  $\quad\{p \wedge j > 0\}\ \{p[s+j/s][j-1/j]\}$
  $\quad\texttt{j := j-1;}\ \{p[s+j/s]\}$
  $\quad\texttt{s := s+j}\ \{p\}$
  od
  $\{p \wedge j \le 0\}\ \{s = sum(0, n)\}$

  is $\{n \ge 0\}\ \texttt{j := n; s := n;}$
  $\{$**inv** $p \equiv 0 \le j \le n \wedge s = sum(j, n)\}$
  while $j > 0$ **do**
  $\quad\texttt{j := j-1; s := s+j}$
  od
  $\{s = sum(0, n)\}$

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 $sp(...)$, and preconditions are inferred from postconditions using $wp(...)$. 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 **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 **if**-**else**:
   Turn $\{ p \}$ **if** B **then** S_1 **else** S_2 **fi** into $\{ p \}$ **if** B **then** $\{p \wedge B\}\ S_1$ **else** $\{ p \wedge \neg B\}\ S_2$ **fi**
5. Add a precondition to an **if**-**else**:
   Prepend $(B \rightarrow p_1) \wedge (\neg B \rightarrow p_2)$ to **if** B **then** $\{p_1\}\ S_1$ **else** $\{p_2\}\ S_2$ **fi**

Add a postcondition:

6. Append $sp(p, v := e)$ to $\{ p \}\ v := e$.
7. Append p to $\{p\}$ **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 **if** B **then** S_1 **else** S_2 **fi** $\{ q \}$ into **if** B **then** $S_1 \{ q \}$ **else** $S_2 \{ q \}$ **fi** $\{ q \}$
   Or turn **if** B **then** S_1 **else** S_2 **fi** $\{ q_1 \vee q_2\}$ into
   if B **then** $S_1 \{ q_1\}$ **else** $S_2 \{ q_2\}$ **fi** $\{ q_1 \vee q_2\}$
10. Add a postcondition to a conditional statement
    Append $q_1 \vee q_2$ to **if** B **then** $S_1 \{ q_1 \}$ **else** $S_2 \{ q_2 \}$ **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 $\{$**inv** $p\}$ **while** B **do** S_1 **od** into $\{$**inv** $p\}$ **while** B **do** $\{ p \wedge B \}\ S_1 \{ p \}$ **od** $\{ p \wedge \neg B\}$

Weaken or strengthen some condition:

12. Turn $\ldots\{ p \}\ldots$ into $\ldots\{ p \}\ \{ q \}\ldots$ for some predicate q where $p \rightarrow q$.
13. Turn $\ldots\{ q \}\ldots$ into $\ldots\{ p \}\ \{ q \}\ldots$ 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;`
  $\{\textbf{inv } p \equiv 0 \leq j \leq n \land s = sum(j, n)\}$
  while $j > 0$ **do**
  　　`j := j-1;`
  　　`s := s+j`
  od
  $\{s = sum(0, n)\}$

- 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;` $\{n \geq 0 \land j = n \land s = n\}$
  $\{\textbf{inv } p \equiv 0 \leq j \leq n \land s = sum(j, n)\}$
  while $j > 0$ **do**
  　　`j := j-1;`
  　　`s := s+j`
  od
  $\{s = sum(0, n)\}$

- 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;` $\{n \geq 0 \land j = n \land s = n\}$
  $\{\textbf{inv } p \equiv 0 \leq j \leq n \land s = sum(j, n)\}$
  while $j > 0$ **do**
  　　$\{p \land j > 0\}$
  　　`j := j-1;`
  　　`s := s+j` $\{p\}$
  od
  $\{p \land j \leq 0\}$ $\{s = sum(0, n)\}$

- 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;` $\{n \geq 0 \land j = n \land s = n\}$
  $\{\textbf{inv } p \equiv 0 \leq j \leq n \land s = sum(j, n)\}$
  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
  $\{p \land j \leq 0\}$ $\{s = sum(0, n)\}$

- 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` $\{p \equiv 0 \leq j \leq n \wedge s = sum(j, n)\}$

  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

    $\{n \geq 0\}$
    $\{0 \leq n \leq n \wedge n = sum(n, n)\}$ `j := n;`
    $\{0 \leq j \leq n \wedge n = sum(j, n)\}$ `s := n` $\{0 \leq j \leq n \wedge s = sum(j, n)\}$

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

    $\{n \geq 0\}$ `j := n;` $\{n \geq 0 \wedge j = n\}$ $\{0 \leq j \leq n \wedge n = sum(j, n)\}$ `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*: $n \geq 0 \wedge j = n \wedge s = n \rightarrow 0 \leq j \leq n \wedge s = sum(j, n)$
    - *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 conditionals $\{p\}$ **if** B **then** $\{p_1\}$ S_1 **else** $\{p_2\}$ S_2 **fi** can become
    - $\{p\}$ **if** B **then** $\{p \wedge B\}\{p_1\}$ S_1 **else** $\{p \wedge \neg B\}$ $\{p_2\}$ S_2 **fi** via case 4 or
    - $\{p\}$ $\{(B \rightarrow p_1) \wedge (\neg B \rightarrow p_2)\}$ **if** B **then** $\{p_1\}$ S_1 **else** $\{p_2\}$ S_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.

Formal Proofs of Partial Correctness and Proof Outlines

CS 536: Science of Programming, Fall 2019

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 partial correctness.

C. Problems

1. The formal proof below is incomplete; fill in the missing rule names (and line references, where needed).

1.	$T \rightarrow 0 \geq 0 \wedge 1 = 2\hat{}0$	________
2.	$\{0 \geq 0 \wedge 1 = 2\hat{}0\}$ i := 0 $\{i \geq 0 \wedge 1 = 2\hat{}i\}$	________
3.	$\{T\}$ i := 0 $\{i \geq 0 \wedge 1 = 2\hat{}i\}$	________
4.	$\{i \geq 0 \wedge 1 = 2\hat{}i\}$ x := 1 $\{i \geq 0 \wedge x = 2\hat{}i\}$	________
5.	$\{T\}$ i := 0; x := 1 $\{i \geq 0 \wedge x = 2\hat{}i\}$	________

Here's an alternate version of the proof that uses forward assignments:

1.	$\{T\}$ i := 0 $\{i = 0\}$	________
2.	$\{i = 0\}$ x := 1 $\{i = 0 \wedge x = 1\}$	________
3.	$\{T\}$ i := 0; x := 1 $\{i = 0 \wedge x = 1\}$	________
4.	$i = 0 \wedge x = 1 \rightarrow i \geq 0 \wedge x = 2\hat{}i$	________
5.	$\{T\}$ i := 0; x := 1 $\{i \geq 0 \wedge x = 2\hat{}i\}$	________

2. Repeat Problem 1 on the incomplete proof below.

1.	$\{-x = abs(x)\}$ y := −x $\{y = abs(x)\}$	________
2.	$y = x \wedge x < 0 \rightarrow -x = abs(x)$	________
3.	$\{y = x \wedge x < 0\}$ y := −x $\{y = abs(x)\}$	________
4.	$\{y = abs(x)\}$ **skip** $\{y = abs(x)\}$	________
5.	$y = x \wedge x \geq 0 \rightarrow y = abs(x)$	________
6.	$\{y = x \wedge x \geq 0\}$ **skip** $\{y = abs(x)\}$	________
7.	$\{y = x\}$ **if** x < 0 **then** y := −x **fi** $\{y = abs(x)\}$	________

3. Repeat Problem 1 on the incomplete proof below.

Below, let $W \equiv$ **while** j > 0 **do** j := j−1; s := s+j **od**

1.	{n ≥ 0} j := n {n ≥ 0 ∧ j = n}	________
2.	{n ≥ 0 ∧ j = n} s := n {n ≥ 0 ∧ j = n ∧ s = n}	________
3.	{n ≥ 0} j := n; s := n {n ≥ 0 ∧ j = n ∧ s = n}	________
4.	n ≥ 0 ∧ j = n ∧ s = n → *p*	________
5.	{n ≥ 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* ∧ j > 0 → *p*[s+j/s][j−1/j]	________
9.	{*p* ∧ j > 0} j := j−1 {*p*[s+j/s]}	________
10.	{*p* ∧ j > 0} j := j−1; s := s+j {*p*}	________
11.	{**inv** *p*} *W* {*p* ∧ j ≤ 0}	________
12.	*p* ∧ j ≤ 0 → s = sum(0, n)	________
13.	{**inv** *p*} *W* {s = sum(0, n)}	________
14.	{n ≥ 0} j := n; s := n; {**inv** *p*} *W* {s = sum(0, n)}	________

4. Write a formal proof of partial correctness for {n > 1} x := n; x := x*x {x ≥ 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 partial correctness from it. There are multiple right answers.

7. {T} {$0 \geq 0 \wedge 1 = 2^0$} i := 0; {$i \geq 0 \wedge 1 = 2^i$} x := 1 {$i \geq 0 \wedge x = 2^i$}

8a. {y = x} **if** x < 0 **then**

    {y = x ∧ x < 0} {−x = abs(x)} y := −x {y = abs(x)}

else

    {y = x ∧ x ≥ 0} {y = abs(x)} **skip** {y = abs(x)}

fi {y = abs(x)}

8b. {y = x} **if** x < 0 **then**

    {y = x ∧ x < 0} y := −x {y_0 = x ∧ x < 0 ∧ y = −x}

else

    {y = x ∧ x ≥ 0} **skip** {y = x ∧ x ≥ 0}

fi {(y_0 = x ∧ x < 0 ∧ y = −x) ∨ (y = x ∧ x ≥ 0)}{y = abs(x)}

8c. $\{y = x\}$ $\{(x < 0 \to -x = abs(x)) \wedge (x \geq 0 \to y = abs(x))\}$

if $x < 0$ **then**

$\{-x = abs(x)\}$ y := -x $\{y = abs(x)\}$

else

$\{y = abs(x)\}$ **skip** $\{y = abs(x)\}$

fi $\{y = abs(x)\}$

9. Hint: Use *sp* for the two loop initialization assignments.

$\{n \geq 0\}$ j := n; $\{n \geq 0 \wedge j = n\}$ s := n; $\{n \geq 0 \wedge j = n \wedge s = n\}$

$\{$**inv** $p \equiv 0 \leq j \leq n \wedge s = sum(j, n)\}$

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

$\{p \wedge j \leq 0\}$ $\{s = sum(0, n)\}$

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 partial correctness. There may be multiple right answers; any right answer is sufficient.

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

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

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.

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

while $i \leq n$ **do**

i := i+1;

s := s+i

od

$\{s = sum(0, n)\}$

Solution to Activities 16 and 17 (Formal Proofs of Partial Correctness and Proof Outlines)

1. Proof:

1.	`T` $\rightarrow$ `0 ≥ 0 ∧ 1 = 2^0`	Predicate logic
2.	`{0 ≥ 0 ∧ 1 = 2^0} i := 0 {i ≥ 0 ∧ 1 = 2^i}`	Assignment (backward)
3.	`{T} i := 0 {i ≥ 0 ∧ 1 = 2^i}`	Precond strengthening 1, 2
4.	`{i ≥ 0 ∧ 1 = 2^i} x := 1 {i ≥ 0 ∧ x = 2^i}`	Assignment (backward)
5.	`{T} i := 0; x := 1 {i ≥ 0 ∧ x = 2^i}`	Sequence 3, 4
	[Alternate version]	
1.	`{T} i := 0 {i = 0}`	Assignment (forward)
2.	`{i = 0} x := 1 {i = 0 ∧ x = 1}`	Assignment (forward)
3.	`{T} i := 0; x := 1 {i = 0 ∧ x = 1}`	Sequence 1, 2
4.	`i = 0 ∧ x = 1 → i ≥ 0 ∧ x = 2^i`	Predicate logic
5.	`{T} i := 0; x := 1 {i ≥ 0 ∧ x = 2^i}`	Postcond weakening 3, 4

2. Proof:

1.	`{-x = abs(x)} y := -x {y = abs(x)}`	Assignment
2.	`y = x ∧ x < 0 → -x = abs(x)`	Predicate logic
3.	`{y = x ∧ x < 0} y := -x {y = abs(x)}`	Precond str 2, 1
4.	`{y = abs(x)}` **`skip`** `{y = abs(x)}`	Skip
5.	`y = x ∧ x ≥ 0 → y = abs(x)}`	Predicate logic
6.	`{y = x ∧ x ≥ 0}` **`skip`** `{y = abs(x)}`	Precond str 5, 4
7.	`{y = x}` **`if`** `x < 0` **`then`** `y := -x` **`fi`** `{y = abs(x)}`	Conditional 3, 6

3. Below, $W \equiv$ **`while`** `j > 0` **`do`** `j:= j-1; s:= s+j` **`od`**

1.	`{n ≥ 0} j := n {n ≥ 0 ∧ j = n}`	Assignment
2.	`{n ≥ 0 ∧ j = n} s := n {n ≥ 0 ∧ j = n ∧ s = n}`	Assignment
3.	`{n ≥ 0} j := n; s := n {n ≥ 0 ∧ j = n ∧ s = n}`	Sequence 1, 2
4.	`n ≥ 0 ∧ j = n ∧ s = n →` *p*	Predicate logic
5.	`{n ≥ 0} j := n; s := n {`*p*`}`	Postcond. weak 3, 4
6.	`{`*p*`[s+j/s]} s:= s+j {`*p*`}`	Assignment
7.	`{`*p*`[s+j/s][j-1/j]} j:= j-1 {`*p*`[s+j/s]}`	Assignment
8.	*p* `∧ j > 0 →` *p*`[s+j/s][j-1/j]`	Predicate logic
9.	`{`*p* `∧ j > 0} j:= j-1 {`*p*`[s+j/s]}`	Precond. str 8, 7
10.	`{`*p* `∧ j > 0} j:= j-1; s:= s+j {`*p*`}`	Sequence 9, 6
11.	`{`**`inv`** *p*`}` *W* `{`*p* `∧ j ≤ 0}`	While 10
12.	*p* `∧ j ≤ 0 → s = sum(0, n)`	Predicate logic
13.	`{`**`inv`** *p*`}` *W* `{s = sum(0, n)}`	Postcond. weak 12, 11
14.	`{n ≥ 0} j := n; s := n; {`**`inv`** *p*`}` *W* `{s = sum(0, n)}`	Sequence 5, 13

4. Proof using *wp*: [modified 10/27]

1.	$\{x*x \ge 4\}\ x := x*x\ \{x \ge 4\}$	(Backward) Assignment
2.	$\{n*n \ge 4\}\ x := n;\ \{x*x \ge 4\}$	(Backward) Assignment
3.	$\{n*n \ge 4\}\ x := n;\ x := x*x\ \{x \ge 4\}$	Sequence 2, 1
4.	$n > 1 \to n*n \ge 4$	Predicate logic
5.	$\{n > 1\}\ x := n;\ x := x*x\ \{x \ge 4\}$	Precond. str. 4, 3

5. Proof using *sp*:

1.	$\{n > 1\}\ x := n;\ \{n > 1 \wedge x = n\}$	(Forward) Assignment
2.	$\{n > 1 \wedge x = n\}\ x := x*x\ \{n > 1 \wedge x_0 = n \wedge x = x_0*x_0\}$	(Forward) Assignment
3.	$\{n > 1\}\ x := n;\ x := x*x\ \{n > 1 \wedge x_0 = n \wedge x = x_0*x_0\}$	Sequence 1, 2
4.	$n > 1 \wedge x_0 = n \wedge x = x_0*x_0 \to x \ge 4$	Predicate logic
5.	$\{n > 1\}\ x := n;\ x := x*x\ \{x \ge 4\}$	Postcond. weak 3, 4

7. (Full outline to proof):

1	$T \to 0 \ge 0 \wedge 1 = 2^0$	Predicate logic
2	$\{0 \ge 0 \wedge 1 = 2^0\}\ i := 0;\ \{i \ge 0 \wedge 1 = 2^i\}$	Assignment
3	$\{T\}\ i := 0;\ \{i \ge 0 \wedge 1 = 2^i\}$	Precond str 1, 2
4	$\{i \ge 0 \wedge 1 = 2^i\}\ x := 1\ \{i \ge 0 \wedge x = 2^i\}$	Assignment
5	$\{T\}\ i := 0;\ x := 1\ \{i \ge 0 \wedge x = 2^i\}$	Sequence 3, 4

8a. (Full outline to proof):

1	$\{-x = abs(x)\}\ y := -x\ \{y = abs(x)\}$	Assignment
2	$y = x \wedge x < 0 \to -x = abs(x)$	Predicate logic
3	$\{y = x \wedge x < 0\}\ y := -x\ \{y = abs(x)\}$	Precond str 2, 1
4	$\{y = abs(x)\}$ **skip** $\{y = abs(x)\}$	Skip
5	$y = x \wedge x \ge 0 \to y = abs(x)\}$	Predicate logic
6	$\{y = x \wedge x \ge 0\}$ **skip** $\{y = abs(x)\}$	Precond str 5, 4
7	$\{y = x\}$ **if** $x < 0$ **then** $y := -x$ **fi** $\{y = abs(x)\}$	Conditional 3, 6

8b. (Full outline to proof):

1	$\{y = x \wedge x < 0\}\ y := -x\ \{y_0 = x \wedge x < 0 \wedge y = -x\}$	Assignment
2	$\{y = x \wedge x \ge 0\}$ **skip** $\{y = x \wedge x \ge 0\}$	Assignment
3	$\{y = x\}$ **if** $x < 0$ **then** $y := -x$ **fi** $\{(y_0 = x \wedge x < 0 \wedge y = -x) \vee (y = x \wedge x \ge 0)\}$	Conditional 1, 2
4	$(y_0 = x \wedge x < 0 \wedge y = -x) \vee (y = x \wedge x \ge 0) \to y = abs(x)$	Predicate logic
5	$\{y = x\}$ **if** $x < 0$ **then** $y := -x$ **fi** $\{y = abs(x)\}$	Postcond. weak., 3, 4

8c. (Full outline to proof):

1	$\{\texttt{-x} = \texttt{abs(x)}\}$ `y := -x` $\{\texttt{y} = \texttt{abs(x)}\}$	Assignment
2	$\{\texttt{y} = \texttt{abs(x)}\}$ **`skip`** $\{\texttt{y} = \texttt{abs(x)}\}$	Skip
3.	$\{p\}$ **`if`** `x < 0` **`then`** `y := -x` **`fi`** $\{\texttt{y} = \texttt{abs(x)}\}$	Conditional 1, 2
	where $p \equiv (\texttt{x} < 0 \to \texttt{-x} = \texttt{abs(x)}) \wedge (\texttt{x} \geq 0 \to \texttt{y} = \texttt{abs(x)})$	
4.	$\texttt{y} = \texttt{x} \to p$	Predicate Logic
5.	$\{\texttt{y} = \texttt{x}\}$ **`if`** `x < 0` **`then`** `y := -x` **`fi`**	
	$\{\texttt{y} = \texttt{abs(x)}\}$	Precond. str. 4, 3

9. Below, let $W \equiv$ **`while`** `j > 0` **`do`** `j := j-1; s := s+j` **`od`**

1	$\{\texttt{n} \geq 0\}$ `j := n` $\{\texttt{n} \geq 0 \wedge \texttt{j} = \texttt{n}\}$	Assignment
2	$\{\texttt{n} \geq 0 \wedge \texttt{j} = \texttt{n}\}$ `s := n`	
	$\{\texttt{n} \geq 0 \wedge \texttt{j} = \texttt{n} \wedge \texttt{s} = \texttt{n}\}$	Assignment
3	$\{\texttt{n} \geq 0\}$ `j := n; s := n`	
	$\{\texttt{n} \geq 0 \wedge \texttt{j} = \texttt{n} \wedge \texttt{s} = \texttt{n}\}$	Sequence 1, 2
4	$\texttt{n} \geq 0 \wedge \texttt{j} = \texttt{n} \wedge \texttt{s} = \texttt{n} \to p$	Predicate logic
5	$\{\texttt{n} \geq 0\}$ `j := n; s := n` $\{p\}$	Postcond. weak 3, 4
6	$\{p[\texttt{s+j}/\texttt{s}]\}$ `s := s+j` $\{p\}$	Assignment
7	$\{p[\texttt{s+j}/\texttt{s}][\texttt{j-1}/\texttt{j}]\}$ `j := j-1` $\{p[\texttt{s+j}/\texttt{s}]\}$	Assignment
8	$p \wedge \texttt{j} > 0 \to p[\texttt{s+j}/\texttt{s}][\texttt{j-1}/\texttt{j}]$	Predicate logic
9	$\{p \wedge \texttt{j} > 0\}$ `j := j-1` $\{p[\texttt{s+j}/\texttt{s}]\}$	Precond. str 8, 7
10	$\{p \wedge \texttt{j} > 0\}$ `j := j-1; s := s+j` $\{p\}$	Sequence 9, 6
11	$\{$**`inv`** $p\}$ W $\{p \wedge \texttt{j} \leq 0\}$	while 10
12	$p \wedge \texttt{j} \leq 0 \to \texttt{s} = \texttt{sum}(0, \texttt{n})$	Predicate logic
13	$\{$**`inv`** $p\}$ W $\{\texttt{s} = \texttt{sum}(0, \texttt{n})\}$	Postcond. weak 12, 11
14	$\{\texttt{n} \geq 0\}$ `j := n; s := n;` W $\{\texttt{s} = \texttt{sum}(0, \texttt{n})\}$	Sequence 5, 13

10. One expansion of $\{\texttt{n} > 1\}$ `i := 1; s := 0` $\{0 \leq \texttt{i} < \texttt{n} \wedge \texttt{s} = \texttt{sum}(0, \texttt{i-1})\}$ uses *sp* (i.e., forward assignment) on the two assignments.

$\{\texttt{n} > 1\}$ `i := 1;` $\{\texttt{n} > 1 \wedge \texttt{i} = 1\}$

$\texttt{s} = 0$ $\{\texttt{n} > 1 \wedge \texttt{i} = 1 \wedge \texttt{s} = 0\}$

$\{0 \leq \texttt{i} < \texttt{n} \wedge \texttt{s} = \texttt{sum}(0, \texttt{i-1})\}$

Another expansion uses *wp* on the two assignments:

$\{\texttt{n} > 1\}$

$\{0 \leq 1 < \texttt{n} \wedge 0 = \texttt{sum}(0, 1\texttt{-}1)\}$ `i := 1;`

$\{0 \leq \texttt{i} < \texttt{n} \wedge 0 = \texttt{sum}(0, \texttt{i-1})\}$ `s := 0` $\{0 \leq \texttt{i} < \texttt{n} \wedge \texttt{s} = \texttt{sum}(0, \texttt{i-1})\}$

Not shown: Use *sp* on the first assignment and *wp* on the second.

11. Here's one possible expansion; it uses the strongest postcondition of the **if** statement. (Note we're dropping "$T \wedge$" in several places.)

$\{T\}$
if $x \geq 0$ **then**
$\{x \geq 0\}$ $y := x$ $\{x \geq 0 \wedge y = x\}$
else
$\{x < 0\}$ $y := -x$ $\{x < 0 \wedge y = -x\}$
fi $\{(x \geq 0 \wedge y = x) \vee (x < 0 \wedge y = -x)\}$ $\{y = abs(x)\}$

Another expansion uses **if** test to get the preconditions of the branches:

$\{T\}$
if $x \geq 0$ **then**
$\{x \geq 0\}$ $\{x = abs(x)\}$ $y := x$ $\{y = abs(x)\}$
else
$\{x < 0\}$ $\{-x = abs(x)\}$ $y := -x$ $\{y = abs(x)\}$
fi $\{y = abs(x)\}$

This expansion uses the *wp* of the conditional:

$\{T\}$ $\{(x \geq 0 \rightarrow x = abs(x)) \wedge (x < 0 \rightarrow -x = abs(x))\}$
if $x \geq 0$ **then**
$\{x = abs(x)\}$ $y := x$ $\{y = abs(x)\}$
else
$\{-x = abs(x)\}$ $y := -x$ $\{y = abs(x)\}$
fi $\{y = abs(x)\}$

12. The expansion is straightforward:

$\{$**inv** $p \equiv 0 \leq i \leq n+1 \wedge s = sum(0, i-1)\}$
while $i \leq n$ **do**
$\{p \wedge i \leq n\}$ $\{p[s+i/s][i+1/i]\}$ $i := i+1;$
$\{p[s+i/s]\}$ $s := s+i$ $\{p\}$
od
$\{p \wedge i > n\}$ $\{s = sum(0, n)\}$

The program is not correct because of the predicate obligation $p \wedge i \leq n \rightarrow p[s+i/s][i+1/i]$ expands to an invalid predicate:

$$0 \leq i \leq n+1 \wedge s = sum(0, i-1) \wedge i \leq n \rightarrow 0 \leq i+1 \leq n+1 \wedge s+(i+1) = sum(0, i+1-1)$$

The error is that we need $s+(i+1) = sum(0, i+1-1)$, but this doesn't follow from $s = sum(0, i-1)$. The new value of s is off by one:

$s = sum(0, i-1)$
$\Rightarrow s+i = sum(0, i-1) + i$

$\Rightarrow \texttt{s+i} = \texttt{sum}(0, \texttt{i})$

$\Rightarrow \texttt{s+i+1} = \texttt{sum}(0, \texttt{i}) + 1$

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

$p \wedge \texttt{i} \le \texttt{n} \rightarrow p[\texttt{i+1}/\texttt{i}][\texttt{s+i}/\texttt{s}]$

$\equiv p \wedge \texttt{i} \le \texttt{n} \rightarrow (0 \le \texttt{i+1} \le \texttt{n+1} \wedge \texttt{s} = \texttt{sum}(0, \texttt{i+1-1}))\,[\texttt{s+i}/\texttt{s}]$

$\equiv 0 \le \texttt{i} \le \texttt{n+1} \wedge \texttt{s} = \texttt{sum}(0, \texttt{i-1}) \wedge \texttt{i} \le \texttt{n}$

$\quad \rightarrow (0 \le \texttt{i+1} \le \texttt{n+1} \wedge \texttt{s+i} = \texttt{sum}(0, \texttt{i+1-1}))$

Another fix is to change `s := s+i` t to `s := s+i-1`. Our obligation becomes

$p \wedge \texttt{i} \le \texttt{n} \rightarrow p[\texttt{s+i-1}/\texttt{s}][\texttt{i+1}/\texttt{i}]$

$\equiv p \wedge \texttt{i} \le \texttt{n} \rightarrow (0 \le \texttt{i} \le \texttt{n+1} \wedge \texttt{s+i-1} = \texttt{sum}(0, \texttt{i-1}))\,[\texttt{i+1}/\texttt{i}]$

$\equiv p \wedge \texttt{i} \le \texttt{n} \rightarrow (0 \le \texttt{i+1} \le \texttt{n+1} \wedge \texttt{s+(i+1)-1} = \texttt{sum}(0, \texttt{i+1-1}))$

