# Proof Outlines for Partial Correctness

## Part 2: Partial and Minimal Proof Outlines

## CS 536: Science of Programming, Fall 2022

### 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, and expand your substitutions somewhere (inline or after outline).

1. Expand the following proof outline:

   $\{n > 1\}\ k := 1;\ s := 0\ \{0 \le k < n \wedge s = sum(0, k-1)\}$

   a. Use wp on both assignments.
   b. Use sp on both assignments..
   c. Use sp on the left assignment and wp on the right assignment.

2. Expand the following proof outline:

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

   a. Use sp on both branches and on the ***if-fi*** as a whole.
   b. Use wp on the ***if-fi*** as a whole and on both branches.
   c. Use $(P \wedge B)$ and $(P \wedge \neg B)$ (from the conditional rule) as overall preconditions for the two branches, and use wp on both branches.

 

3. Expand the proof outline below.
   a. Use wp everywhere you can.
   b. Use sp everywhere you can.

```
{n ≥ 0}
x := 0;
y := 1;
{inv P(a, b, x, y)}
while x < n do
    x := f(x, y);
    y := f(y, x)
od
{a + x < b – y}
```

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.

```
{inv p ≡ 0 ≤ k ≤ n+1 ∧ s = sum(0, k–1)}
while k ≤ n do
    k := k+1;
    s := s+k
od
{s = sum(0, n)}
```

 

## *Solution to Practice 17 (Partial 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 ≤ 1 < n ∧ 0 = sum(0, 1–1)} k := 1;
{0 ≤ k < n ∧ 0 = sum(0, k–1)} s := 0 {0 ≤ k < n ∧ s = sum(0, k–1)}

The predicate logic obligation is n > 1 → (0 ≤ 1 < n ∧ 0 = sum(0, 1–1))

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

{n > 1}
k := 1; {n > 1 ∧ k = 1}
s = 0 {n > 1 ∧ k = 1 ∧ s = 0}
{0 ≤ k < n ∧ s = sum(0, k–1)}

The predicate logic obligation is (n > 1 ∧ k = 1 ∧ s = 0) → (0 ≤ k < n ∧ s = sum(0, k–1))

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

{n > 1}
k := 1; {n > 1 ∧ k = 1}
{0 ≤ k < n ∧ 0 = sum(0, k–1)} s := 0
{0 ≤ k < n ∧ s = sum(0, k–1)}

The predicate logic obligation is (n > 1 ∧ k = 1) → (0 ≤ k < n ∧ 0 = sum(0, k–1)).

2. (Expansions of minimal outline for an ***if-else***.) Note "T ∧" has been dropped in various places.

2a. An expansion using sp on both branches and on the whole ***if-else:***

{T}
***if*** x ≥ 0 ***then***
    {x ≥ 0} y := x {x ≥ 0 ∧ y = x}
***else***
    {x < 0} y := –x {x < 0 ∧ y = –x}
***fi***
{(x ≥ 0 ∧ y = x) ∨ (x < 0 ∧ y = –x)}
{y = abs(x)}

The predicate logic obligation is ((x ≥ 0 ∧ y = x) ∨ (x < 0 ∧ y = –x)) → y = abs(x).

---

* The decision of where to place line breaks in the full outline is stylistic. Your style can certainly differ, but be consistent in your indentation. E.g., don't indent the true branch of an ***if-else*** but not the false branch.

 

2b. An expansion using wp on the whole if-else and both branches:

{T} {(x ≥ 0 → x = abs(x)) ∧ (x < 0 → –x = abs(x))}
***if*** x ≥ 0 ***then***
    {x = abs(x)} y := x {y = abs(x)}
***else***
    {–x = abs(x)} y := –x {y = abs(x)}
***fi***
{y = abs(x)}

The logic obligation is (T → (x ≥ 0 → x = abs(x)) ∧ (x < 0 → –x = abs(x)))

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

{T}
***if*** x ≥ 0 ***then***
    {x ≥ 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 time there are two logic obligations, namely, the conjuncts from part (b): (x ≥ 0 → x = abs(x)) and (x < 0 → –x = abs(x)).

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

{n ≥ 0} x := 0; y := 1;
{***inv*** P(a, b, x, y)} ***while*** x < n ***do*** x := f(x, y); y := f(y, x) ***od*** {a + x < b – y}

3a. An expansion using wp as much as possible:

{n ≥ 0}
{P(a, b, 0, 1) x := 0;
{P(a, b, x, 1)} y := 1;
{***inv*** P(a, b, x, y)}
***while*** x < n ***do***
    {x < n ∧ 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)}
***od***
{x ≥ n ∧ 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.

$(n \geq 0 \rightarrow P(a, b, 0, 1)$***)***
$(x < n \wedge P(a, b, x, f(y, x))) \rightarrow P(a, b, f(x, y), f(y, f(x, y)))$
$(x \geq n \wedge P(a, b, x, y) \rightarrow a + x < b - y)$

3b. An expansion using sp as much as possible.

$\{n \geq 0\}$ x := 0; $\{n \geq 0 \wedge x = 0\}$
y := 1; $\{n \geq 0 \wedge x = 0 \wedge y = 1\}$
{***inv*** $P(a, b, x, y)\}$
***while*** $x < n$ ***do***
    $\{x < n \wedge P(a, b, x, f(y, x))\}$
    $\{P(a, b, x_0, y_0) \wedge y = f(y_0, x_0) \wedge x = f(x_0, y)\}$
    x := f(x, y);
    $\{P(a, b, x, y_0) \wedge y = f(y_0, x)\}$
    y := f(y, x)
    $\{P(a, b, x, y)\}$
***od***
$\{x \geq n \wedge P(a, b, x, y)\}$
$\{a + x < b - y\}$

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 {***inv*** ...} ***while*** ..., not by the loop body.

$(n \geq 0 \wedge x = 0 \wedge y = 1 \rightarrow P(a, b, x, y)$***)***
$(x < n \wedge P(a, b, x, f(y, x))) \rightarrow P(a, b, x_0, y_0) \wedge y = f(y_0, x_0) \wedge x = f(x_0, y))$
$(x \geq n \wedge P(a, b, x, y) \rightarrow a + x < b - y)$

4. The expansion is straightforward:

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

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

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

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

$s = sum(0, k-1)$

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

$\Rightarrow s+k = sum(0, k)$

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

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

$p \wedge k \leq n \rightarrow p[k+1/k][s+k/s]$

$\equiv p \wedge k \leq n \rightarrow (0 \leq k+1 \leq n+1 \wedge s = sum(0, k+1-1))[s+k/s]$

$\equiv 0 \leq k \leq n+1 \wedge s = sum(0, k-1) \wedge k \leq n$

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

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

$p \wedge k \leq n \rightarrow p[s+k-1/s][k+1/k]$

$\equiv p \wedge k \leq n \rightarrow (0 \leq k \leq n+1 \wedge s+k-1 = sum(0, k-1))[k+1/k]$

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