Proof Outlines

Stefan Muller
based on material by Jim Sasaki

CS 536: Science of Programming, Fall 2023
Lecture 16

1 Full and Minimal Proof Outlines

- When learning about different kinds of proofs, *proof outlines* were generally the most compact, since they avoided writing a lot of redundant conditions.
- But they were also the most mysterious in terms of how to actually write one.
- We also saw that most of the information in a proof outline can actually be computed from weakest preconditions and strongest postconditions.
- We'll use the term *minimal proof outline* to refer to a proof outline that contains only the information that *can't* be computed. Generally, this will be the initial precondition, the final postcondition, and loop invariants. (If there are particularly tricky applications of the Consequence rule, you might make those explicit too.)
- We'll use the term *full proof outline* to refer to the proof outlines we've seen so far: they contain all of the information that a proof tree or Hilbert-style proof has, just without writing redundant text.
- Today, we'll write an explicit algorithm for building a full proof outline from a minimal proof outline.
- In the process, you'll check that the minimum proof outline is correct (e.g., loop invariants are right).
- Why would you want to do this? Two possible reasons:
  - If someone else has proven a piece of code and gives you a minimal proof outline, you might do this to check that their proof was correct.
  - If you're trying to find a loop invariant, a good way to check the three conditions that make a loop invariant correct (that it's true at the start of the loop, that it plus the negation of the loop condition imply the postcondition and that it's maintained by the loop body) is to write a proof outline for at least the part of the program around the loop.

Example 1.
Here's a full proof outline from a previous class, with the removable parts in green.

```
                    $\{n \geq 0\}$                                              // Init. precond. can't be inferred
$k := \overline{0};$             $\{n \geq 0 \wedge k = 0\}$                              // Inferred w/ sp
$s := \overline{0};$             $\{n \geq 0 \wedge k = 0 \wedge s = 0\}$                      // Inferred w/ sp
$\{\textbf{inv } p \triangleq 0 \leq k \leq n \wedge s = sum(0,k)\}$                      // Loop inv. can't be inferred
while $k < n$ do   $\{p \wedge k < n\}$                                         // Loop body gets $p \wedge e$
                    $\Rightarrow \{0 \leq k+1 \leq n \wedge s+k = sum(0,k+1)\}$    // Inferred w/ wp
  $s := s + k$        $\{0 \leq k+1 \leq n \wedge s = sum(0,k+1)\}$              // Inferred w/ wp
  $k := k + \overline{1}$       $\{p\}$                                              // Body must end w/ loop inv.
od                  $\{p \wedge k \geq n\}$                                         // Loop ends w/ $p \wedge \neg e$
                    $\Rightarrow \{s = sum(0,n)\}$                                  // Final postcond. can't be inferred
```

If we drop everything that can be inferred (everything in green), we get the minimal proof outline:

```
                    {$n \geq 0$}
$k := \overline{0}$;
$s := \overline{0}$;
{**inv** $p \triangleq 0 \leq k \leq n \wedge s = sum(0,k)$}
while $k < n$ do
   $s := s + k$
   $k := k + \overline{1}$
od
                    {$s = sum(0,n)$}
```

Example 2. For a given minimal proof outline (precondition, loop invariants, and postcondition), there may be several full proof outlines.

Remember we had two rules for assignment. The "backward rule":

$$\{[e/x]Q\}\ x := e\ \{Q\}$$

and the "forward rule":

$$\{P\}\ x := e\ \{\exists x_0.[x_0/x]P \wedge x = [x_0/x]e\}$$

We use the backward rule to calculate weakest preconditions and the forward rule to calculate strongest postconditons. Sometimes, we don't really have much of a choice which one we want to do, but sometimes we can start at the top and calculate sps, or start at the bottom and calculate wps.

Consider the following minimal proof outline:

$$\{T\}\ k := \overline{0}; x := \overline{1}\ \{k \geq 0 \wedge x = 2^k\}$$

It corresponds to all three of these full proof outlines.

- $\{T\}\ \Rightarrow \{0 \geq 0 \wedge 1 = 2^0\} k := \overline{0}; \{k \geq 0 \wedge 1 = 2^k\} x := \overline{1}\ \{k \geq 0 \wedge x = 2^k\}$
- $\{T\}\ k := \overline{0}; \{k = 0\} x := \overline{1} \{k = 0 \wedge x = 1\} \Rightarrow\ \{k \geq 0 \wedge x = 2^k\}$
- $\{T\}\ k := \overline{0}; \{k = 0\} \Rightarrow \{k \geq 0 \wedge 1 = 2^k\} x := \overline{1}\ \{k \geq 0 \wedge x = 2^k\}$

In the first case, we started at the end and calculated weakest preconditions until we got to the precondition. In the second case, we started at the beginning and calculated strongest postconditions until we got to the final postcondition. In the third case, we calculated a weakest precondition for the second statement and a strongest postcondition for the first.

Note that in every case, when we reached a condition we already had (in the first case, when we reached the precondition, in the second case when we reached the postcondition, and in the third case when the conditions ran into each other in the middle), we needed to show that one condition implied another. This is a *predicate logic proof obligation*: it's something we need to prove in order to use the Consequence rule, and we prove it using the rules of predicate logic (and usually some other domain like arithmetic) rather than the rules of Hoare logic.

2 Expanding Minimal Proof Outlines

- Below, we will introduce an algorithm for inferring the missing conditions (using wp and sp) to turn a minimal proof outline into a full proof outline.
- There will be some choices in the algorithm (the algorithm is fairly informal and *nondeterministic*), and depending on how we make these choices, we may get different full proof outlines from the same minimal proof outline (as in Example 2), but all will be correct.
- The algorithm adds preconditions using the various wlp rules, as well as the preconditions for loop bodies and conditional branches from the Hoare logic rules, and adds postconditions using the various sp rules, as well as adding the postconditions of loops and conditionals.

2.1 Algorithm

Until every statement can be proved by a triple (i.e., we have a full proof outline that can be systematically turned into a Hilbert-style proof), do one of the following:

1. Prepend $\{wlp(x := e, Q)\}$ to $x := e\ \{Q\}$.
2. Prepend $\{Q\}$ to $\mathsf{skip}\ \{Q\}$.
3. Add preconditions to the branches of a conditional:
$$\{P\}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$$
becomes
$$\{P\}\mathsf{if}\ e\ \mathsf{then}\ \{P \wedge e\}S_1\ \mathsf{else}\ \{P \wedge \neg e\}S_2\ \mathsf{fi}$$
4. Propagate preconditions from branches to the start of a conditional:
$$\mathsf{if}\ e\ \mathsf{then}\ \{P_1\}S_1\ \mathsf{else}\ \{P_2\}S_2\ \mathsf{fi}$$
becomes
$$\{(e \rightarrow P_1) \wedge (\neg e \rightarrow P_2)\}\mathsf{if}\ e\ \mathsf{then}\ \{P_1\}S_1\ \mathsf{else}\ \{P_2\}S_2\ \mathsf{fi}$$
5. Append $\{sp(x := e, P)\}$ to $\{P\}\ x := e$
6. Append $\{P\}$ to $\{P\}\ \mathsf{skip}$
7. Propagate postconditions from branches of a conditional:
$$\mathsf{if}\ e\ \mathsf{then}\ S_1\ \{Q_1\}\ \mathsf{else}\ S_2\ \{Q_2\}\ \mathsf{fi}$$
becomes
$$\mathsf{if}\ e\ \mathsf{then}\ S_1\ \{Q_1\}\ \mathsf{else}\ S_2\ \{Q_2\}\ \mathsf{fi}\ \{Q_1 \vee Q_2\}$$
8. Add postconditions to the branches of a conditional:
$$\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\ \{Q\}$$
becomes
$$\mathsf{if}\ e\ \mathsf{then}\ S_1\ \{Q\}\ \mathsf{else}\ S_2\ \{Q\}\ \mathsf{fi}\ \{Q\}$$
9. Add conditions to a loop based on the invariant:
$$\{\mathbf{inv}\ P\}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}$$
becomes
$$\{\mathbf{inv}\ P\}\ \mathsf{while}\ e\ \mathsf{do}\ \{P \wedge e\}\ S\ \{P\}\ \mathsf{od}\ \{P \wedge \neg e\}$$
10. Check/solve a predicate logic proof obligation: If you've placed two conditions next to each other using the above rules, e.g., $\{P\}\{Q\}$, make sure that $P \Rightarrow Q$

Note. There's another important proof obligation that may not be obvious: when we have a loop invariant, it needs to be implied by the condition immediately before, e.g.

$$\begin{array}{ll} S & \{P_1\} \\ \{\mathbf{inv}\ P_2\} & \\ \mathsf{while}\ e\ \mathsf{do} & \ldots \end{array}$$

gives the proof obligation $P_1 \Rightarrow P_2$.

Example 3. Let's take the minimal proof outline from before and expand it into a full proof outline. Remember, there are a few correct ways to do this and, while they might give different proof outlines, they'll still be correct (some may just be easier to complete than others).

```
                  {$n \geq 0$}
$k := \overline{0}$;
$s := \overline{0}$;
{**inv** $p \triangleq 0 \leq k \leq n \wedge s = sum(0,k)$}
while $k < n$ do
   $s := s + k$
   $k := k + \overline{1}$
od
                  {$s = sum(0,k)$}
```

1. Use step 9 to propagate the loop invariant.

```
                   {$n \geq 0$}
$k := \overline{0}$;
$s := \overline{0}$;
{**inv** $p \triangleq 0 \leq k \leq n \wedge s = sum(0,k)$}
while $k < n$ do   {$p \wedge k < n$}
   $s := s + k$
   $k := k + \overline{1}$      {$0 \leq k \leq n \wedge s = sum(0,k)$}
od                 {$p \wedge k \geq n$}
                   {$s = sum(0,n)$}
```

2. We now have a proof obligation:
$$p \wedge k \geq n \Rightarrow s = sum(0,k)$$
that is,
$$0 \leq k \leq n \wedge s = sum(0,k) \wedge k \geq n \Rightarrow s = sum(0,k)$$
so this is a good time to check that it's correct (if not, this probably means our loop invariant wasn't strong enough and we need to go back and fix it). Fortunately, this is the case since $k \leq n \wedge k \geq n \Rightarrow k = n$ and so $s = sum(0,k) \Rightarrow \mathrm{sum}(0,n)$.

3. Now let's use wlp to propagate the postcondition of the loop body backwards (we do this with two applications of step 1).

```
                   {$n \geq 0$}
$k := \overline{0}$;
$s := \overline{0}$;
{**inv** $p \triangleq 0 \leq k \leq n \wedge s = sum(0,k)$}
while $k < n$ do   {$p \wedge k < n$}
                   {$0 \leq k+1 \leq n \wedge s+k = sum(0,k+1)$}
   $s := s + k$       {$0 \leq k+1 \leq n \wedge s = sum(0,k+1)$}
   $k := k + \overline{1}$      {$0 \leq k \leq n \wedge s = sum(0,k)$}
od                 {$p \wedge k \geq n$}
                   $\Rightarrow$ {$s = sum(0,n)$}
```

4. This gives us another proof obligation:
$$0 \leq k \leq n \wedge s = sum(0,k) \wedge k < n \Rightarrow 0 \leq k+1 \leq n \wedge s+k = sum(0,k+1)$$
Again, this is the case since $k < n \Rightarrow k+1 \leq n$ and
$$s = sum(0,k) \Rightarrow s+k = sum(0,k)+k = sum(0,k+1)$$

5. We can do the same thing, propagating the loop invariant back through the assignments at the beginning of the program, using step 1.

$$\begin{array}{ll}
 & \{n \geq 0\} \\
 & \{0 \leq 0 \leq n \land 0 = \mathit{sum}(0,0)\} \\
k := \overline{0}; & \{0 \leq k \leq n \land 0 = \mathit{sum}(0,k)\} \\
s := \overline{0}; & \{0 \leq k \leq n \land s = \mathit{sum}(0,k)\} \\
\multicolumn{2}{l}{\{\textbf{inv } p \triangleq 0 \leq k \leq n \land s = \mathit{sum}(0,k)\}} \\
\mathsf{while}\ k < n\ \mathsf{do} & \{p \land k < n\} \\
 & \Rightarrow \{0 \leq k+1 \leq n \land s + k = \mathit{sum}(0,k+1)\} \\
\quad s := s + k & \{0 \leq k+1 \leq n \land s = \mathit{sum}(0,k+1)\} \\
\quad k := k + \overline{1} & \{0 \leq k \leq n \land s = \mathit{sum}(0,k)\} \\
\mathsf{od} & \{p \land k \geq n\} \\
 & \Rightarrow \{s = \mathit{sum}(0,n)\}
\end{array}$$

(We could also have propagated the precondition $n \geq 0$ forward up to the loop, which would give us the proof obligation that $n \geq 0 \land k = 0 \land s = 0$ implies the loop invariant, which also holds.)

6. This gives us the proof obligation

$$n \geq 0 \Rightarrow 0 \leq 0 \leq n \land 0 = \mathit{sum}(0,0)$$

which is fairly clear.

We now have the completed full proof outline:

$$\begin{array}{ll}
 & \{n \geq 0\} \\
 & \Rightarrow \{0 \leq 0 \leq n \land 0 = \mathit{sum}(0,0)\} \\
k := \overline{0}; & \{0 \leq k \leq n \land 0 = \mathit{sum}(0,k)\} \\
s := \overline{0}; & \{0 \leq k \leq n \land s = \mathit{sum}(0,k)\} \\
\multicolumn{2}{l}{\{\textbf{inv } p \triangleq 0 \leq k \leq n \land s = \mathit{sum}(0,k)\}} \\
\mathsf{while}\ k < n\ \mathsf{do} & \{p \land k < n\} \\
 & \Rightarrow \{0 \leq k+1 \leq n \land s + k = \mathit{sum}(0,k+1)\} \\
\quad s := s + k & \{0 \leq k+1 \leq n \land s = \mathit{sum}(0,k+1)\} \\
\quad k := k + \overline{1} & \{0 \leq k \leq n \land s = \mathit{sum}(0,k)\} \\
\mathsf{od} & \{p \land k \geq n\} \\
 & \Rightarrow \{s = \mathit{sum}(0,n)\}
\end{array}$$

Note that this is different from the full proof outline for this program in Example 1 (which was constructed using sp for the first couple assignments instead of wp), but both are correct.

As we saw in Example 2, using different combinations of wp and sp will generally work, it'll just give different conditions and different proof obligations. Which to use is generally a matter of taste and preference (and sometimes, one or the other will result in slightly nicer and/or easier to prove conditions).