Summing an array

Here we use proof outlines synthetically.

We design a loop ‘invariant first’.

Recall

That $\{p, ..q\} = \{i \mid p \leq i < q\}$.

Hence $\{p, ..p\} = \emptyset$.

And, if $p \leq q$, then $\{p, ..q+1\} = \{p, ..q\} \cup \{q\}$ so that

$$\sum_{i \in \{p,..q+1\}} f(i) = f(q) + \sum_{i \in \{p,..q\}} f(i)$$

The problem

Suppose a is an array of domain $\{0, ..a.\mathrm{length}\}$.

We want s to be set to the sum of the values in the array.

We can specify the problem with a precondition and a postcondition.

$\{ \mathrm{true} \}$
?
$\{ s = \sum_{i \in \{0,..a.\mathrm{length}\}} a(i) \}$

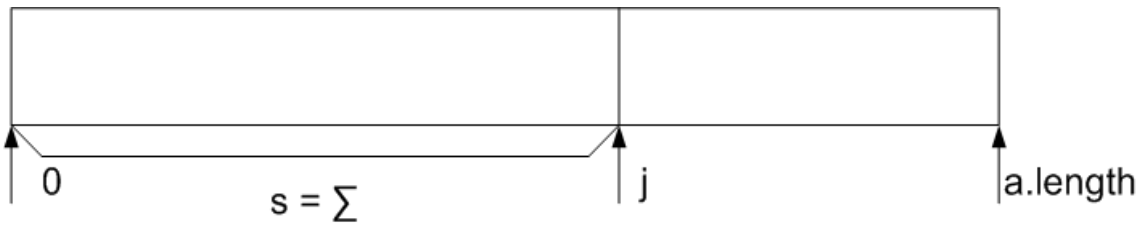
(without changing a)

Creating an invariant

Let's consider 'jumping into the middle'

If at some point we have summed the first j items of the array, where $j \in \{0, .., a.\,\mathrm{length}\}$, what will be true?

In a picture:



As a formula

We can use this break the problem into two problems.

$\{\ \mathrm{true}\ \}$
?
$\{\ \mathcal{I} : 0 \le j \le a.\,\mathrm{length} \wedge s = \sum_{i \in \{0,..j\}} a(i)\ \}$
?
$\{\ s = \sum_{i \in \{0,..a.\,\mathrm{length}\}} a(i)\ \}$

Initialization

The first problem is easily solved

$\{ \text{true} \}$

$\{ \mathcal{I} : 0 \le j \le a.\text{length} \wedge s = \sum_{i \in \{0,..j\}} a(i) \}$

since the VC $\text{true} \Rightarrow \mathcal{I}[s : 0][j : 0]$ is universally true

Iteration

The remaining problem is

$\{ \mathcal{I} : 0 \le j \le a.\text{length} \wedge s = \sum_{i \in \{0,..j\}} a(i) \}$

?

$\{ s = \sum_{i \in \{0,..a.\text{length}\}} a(i) \}$

If we could somehow make $j = a.\text{length}$ true, as well as $\mathcal{I}$, we'd be done.

But this is what a while loop does if $\mathcal{I}$ is its invariant and $j \neq a.\text{length}$ is its guard.

We have:

$\{ \mathcal{I} : 0 \le j \le a.\text{length} \wedge s = \sum_{i \in \{0,..j\}} a(i) \}$

while $G : j \neq a.\text{length}$ do

  $\{ \mathcal{I} \wedge \mathcal{G} \}$

  ?

end while

$\{ \mathcal{R} : s = \sum_{i \in \{0,..a.\text{length}\}} a(i) \}$

since $\mathcal{I} \wedge \neg\mathcal{G} \Rightarrow \mathcal{R}$ is universally true.

The body

It only remains to find a loop body

$\{\ \mathcal{I} \wedge \mathcal{G} : 0 \le j < a.\mathrm{length} \wedge s = \sum_{i \in \{0,..j\}} a(i)\ \}$

?

$\{\ \mathcal{I} : 0 \le j \le a.\mathrm{length} \wedge s = \sum_{i \in \{0,..j\}} a(i)\ \}$

We want each iteration of the loop to get closer to $j = a.\mathrm{length}$, so it makes sense to increment j and to make a corresponding adjustment to s to enure the invariant true at the end of each iteration.

This leads to

$\{\ \mathcal{I} \wedge \mathcal{G} : 0 \le j < a.\mathrm{length} \wedge s = \sum_{i \in \{0,..j\}} a(i)\ \}$

$\{\ \mathcal{I} : 0 \le j \le a.\mathrm{length} \wedge s = \sum_{i \in \{0,..j\}} a(i)\ \}$

By the way, because of the index operation, we should also check that the precondition of $s := s + a(j)$ implies that $0 \le j < a.\mathrm{length}$, which it does.

Now we need to verify that the body re-establishes the invariant. We need to check that

$\mathcal{I}$ is $0 \le j \le a.\text{length} \wedge s = \sum_{i \in \{0,..j\}} a(i)$ and
$\mathcal{G}$ is $j \ne a.\text{length}$.

To verify, make the substitutions and check that the precondition implies the substituted postcondition.

$$\begin{array}{ll}
& \mathcal{I}[j : j+1][s : s + a(j)] \\
= & \quad \text{substitute} \\
& \left(0 \le j+1 \le a.\text{length} \wedge s = \sum_{i \in \{0,..j+1\}} a(i) \right) [s : s + a(j)] \\
= & \quad \text{substitute} \\
& \left(0 \le j+1 \le a.\text{length} \wedge s + a(j) = \sum_{i \in \{0,..j+1\}} a(i) \right) \\
\Leftarrow & \quad \text{since } 0 \le j \Rightarrow 0 \le j+1 \text{ and } (j+1 < n) = (j \le n) \\
& \left(0 \le j < a.\text{length} \wedge s + a(j) = \sum_{i \in \{0,..j+1\}} a(i) \right) \\
= & \quad \text{split the summation} \\
& \left(0 \le j < a.\text{length} \wedge s + a(j) = a(j) + \sum_{i \in \{0,..j\}} a(i) \right) \\
= & \quad \text{cancel the } a(j)\text{s} \\
& 0 \le j < a.\text{length} \wedge s = \sum_{i \in \{0,..j\}} a(i) \\
= & \quad \text{definitions of } \mathcal{G} \text{ and } \mathcal{I} \\
& \mathcal{G} \wedge \mathcal{I}
\end{array}$$

And this is exactly the precondition of the loop body

Summary

We have developed and verified the following proof outline

$\{ \text{true} \}$
$j := 0$
$s := 0$
$\{ \mathcal{I} : 0 \leq j \leq a.\text{length} \wedge s = \sum_{i \in \{0,..j\}} a(i) \}$
while $\mathcal{G} : j \neq a.\text{length}$ **do**
  $\{ \mathcal{I} \wedge \mathcal{G} \}$
  $s := s + a(j)$
  $j := j + 1$
end while
$\{ s = \sum_{i \in \{0,..a.\text{length}\}} a(i) \}$

Reflection

You can use proof outlines and the programming rules in several ways

- Synthetically. I.e.: top-down step-wise refinement.

  This is what we did above. First we found the required assertions (especially the loop invariant) and then filled in the program to make a correct outline
- Analytically: I.e.: post-facto verification

  Start with the algorithm and try to find assertions that make a correct proof outline.
- Review:

  Start with proof outline and show it is correct.

Generally we combine the approaches. But it is worthwhile to try to take a synthetic approach as much as possible.

In this way, we use our desire to produce a correct program to steer the development.

On proofs of programs vs. proofs of (other) mathematical theorems

Mathematical theorems are often conjectured based on observation or need.

- Thus the statement of the theorem often provides little indication of how to prove it.

Examples: Fermat's last theorem and the 4-colour theorem.

Programs are generally written by programmers who have a reason for believing their program will work.

- A proof outline provides a written record of that reasoning.

It is not much harder to construct a proof outline than to construct a program.

In fact, because writing down our reasons often helps to clarify them, writing a proof outline may be easier than simply writing the unannotated program.

In Engineering, theorems are important if they have implications for the soundness of a design. Thus in Engineering there is no shame in stating and proving easy theorems.

Even if you are not trying to create a rigorous proof, it is still important, while creating a program, to think about and document your reasons for believing that it is correct.

The ‘physics’ of programming

What Galileo and Newton did was to invent methods to reduce question of physics (and physical engineering) to questions of “ordinary mathematics”. An important component of this is to tame time by treating it as a real variable.

Proof outlines provide a bridge that reduces questions of computer engineering to questions of ordinary mathematics. Proof outlines tame time in a different way: by asking what must be true before an action to ensure that something else must be true after the action.
