Programming Languages: Imperative Program Construction
Practicals 6: Loop Constuction II

Shin-Cheng Mu

Autumn Term, 2024

1. Recall the maximum segment sum problem. What if we want to compute the maximum sum of *non-empty* segments?

   (a) How would you write the specification? Does the specification still make sense with N being constrained only by $0 \leqslant N$?

   > **Solution:** The specification could be:
   >
   > $$\begin{array}{l} \textbf{con } N : \mathit{Int}\ \{0 \leqslant N\} \\ \textbf{con } f : \textbf{array } [0..N) \textbf{ of } \mathit{Int} \\ S \\ \{r = \langle \uparrow p\ q : 0 \leqslant p < q \leqslant N : \mathit{sum}\ p\ q\rangle\}\ \ , \end{array}$$
   >
   > where the definition of *sum* is unchanged:
   >
   > $$\mathit{sum}\ p\ q = \langle \Sigma i : p \leqslant i < q : f[i]\rangle\ \ .$$
   >
   > When $N = 0$, $0 \leqslant p < q \leqslant N$ reduces to *False* and r should be $-\infty$. The specification is fine if $-\infty$ is a value allowed in our program.

   (b) Derive a program solving the problem, assuming $-\infty$ to be the identity of $(\uparrow)$.

   > **Solution:** Like in the handouts, we start with
   >
   > $$\begin{array}{l} P_0 \equiv r = \langle \uparrow p\ q : 0 \leqslant p < q \leqslant n : \mathit{sum}\ p\ q\rangle\ \ , \\ Q \equiv 0 \leqslant n \leqslant N\ \ , \end{array}$$
   >
   > and use $N - n$ as bound.
   > To find out what we can do with r before $n := n + 1$, we calculate:
   >
   > $$\begin{array}{ll} & \langle \uparrow p\ q : 0 \leqslant p < q \leqslant n : \mathit{sum}\ p\ q\rangle[n\backslash n+1] \\ = & \langle \uparrow p\ q : 0 \leqslant p < q \leqslant n+1 : \mathit{sum}\ p\ q\rangle \\ = & \quad \{\text{ split off } q = n+1 \text{ (safe since } 0 \leqslant n) \} \\ & \langle \uparrow p\ q : 0 \leqslant p < q \leqslant n : \mathit{sum}\ p\ q\rangle \uparrow \langle \uparrow p : 0 \leqslant p < n+1 : \mathit{sum}\ p\ (n+1)\rangle\ \ . \end{array}$$
   >
   > Therefore we add another invariant:
   >
   > $$P_1 \equiv s = \langle \uparrow p : 0 \leqslant p < n : \mathit{sum}\ p\ n\rangle\ \ .$$
   >
   > Variables r, s, n can be initialised by $-\infty, -\infty, 0$. To find out how to update s, we calculate:

$$
\begin{array}{ll}
& \langle \uparrow p : 0 \leqslant p < n : sum\ p\ n \rangle [n \backslash n+1] \\
= & \langle \uparrow p : 0 \leqslant p < n+1 : sum\ p\ (n+1) \rangle \\
= & \quad \{ \text{ split off } p = n \text{ (safe since } 0 \leqslant n) \ \} \\
& \langle \uparrow p : 0 \leqslant p < n : sum\ p\ (n+1) \rangle \uparrow sum\ n\ (n+1) \\
= & \quad \{ \text{ definition of } sum\text{, one-point rule } \} \\
& \langle \uparrow p : 0 \leqslant p < n : sum\ p\ (n+1) \rangle \uparrow f[n] \\
= & \quad \{ \text{ split off } i = n \text{ in definition of } sum\text{, (safe since } 0 \leqslant n) \ \} \\
& \langle \uparrow p : 0 \leqslant p < n : sum\ p\ n + f[n] \rangle \uparrow f[n] \\
= & \quad \{\ p \text{ not free in } f[n] \ \} \\
& (\langle \uparrow p : 0 \leqslant p < n : sum\ p\ n \rangle + f[n]) \uparrow f[n] \quad .
\end{array}
$$

The constructed program is:

con $N : Int\ \{0 \leqslant N\}$
con f : **array** $[0..N)$ **of** Int
var $r, s, n : Int$

$r, s, n := -\infty, -\infty, 0$
$\{P_0 \wedge P_1 \wedge Q, bnd : N - n\}$
do $n \neq N \rightarrow$
  $s := (s + f[n]) \uparrow f[n]$
  $r := r \uparrow s$
  $n := n + 1$
od
$\{r = \langle \uparrow p\ q : 0 \leqslant p < q \leqslant N : sum\ p\ q \rangle\}$

2. Let $n : Nat$. Trying explaining why the following "splitting off" step is wrong by trying out the range calculation.

$$
\begin{array}{ll}
& \langle \Sigma i : n+1 \leqslant i < n+1 : f[i] \rangle \\
= & \quad \{ \text{ splitting off } i = n \ \} \\
& \langle \Sigma i : n+1 \leqslant i < n : f[i] \rangle + f[n] \quad .
\end{array}
$$

Solution: Let's calculate:

$$
\begin{array}{ll}
& n+1 \leqslant i < n+1 \\
= & n+1 \leqslant i \wedge (i < n \vee i = n) \\
= & n+1 \leqslant i < n \vee (n+1 \leqslant i \wedge i = n) \\
= & n+1 \leqslant i < n \vee False \\
= & n+1 \leqslant i < n \quad .
\end{array}
$$

The term $n+1 \leqslant i \wedge i = n$ simplifies to $False$, not $i = n$, which the "splitting off" step might expect.

What we do have is

$$
\begin{array}{ll}
& \langle \Sigma i : n+1 \leqslant i < n+1 : f[i] \rangle \\
= & \langle \Sigma i : n+1 \leqslant i < n : f[i] \rangle \quad .
\end{array}
$$

In fact, both expressions evalulate to 0.

3. Derive a solution for:

$$\begin{array}{l} \textbf{con } N : Int\{N \geqslant 0\}; a : \textbf{array } [0..N) \textbf{ of } Int \\ \textbf{var } r : Int \\ S \\ \{r = \langle \uparrow i, j : 0 \leqslant i < j < N : a[i] - a[j] \rangle\} \ . \end{array}$$

Solution: Replace constant N by variable n, and use a loop that increments n in each step. Use the following P as a candidate of the loop invariant

$$P \equiv r = \langle \uparrow i, j : 0 \leqslant i < j < n : a[i] - a[j] \rangle.$$

To find out how to update r such that we may increment n, we calculate (assuming $0 \leqslant n$):

$$\begin{array}{ll} & \langle \uparrow i, j : 0 \leqslant i < j < n + 1 : a[i] - a[j] \rangle \\ = & \{ \text{ since } 0 \leqslant n, \text{ split off } j = n \ \} \\ & \langle \uparrow i, j : 0 \leqslant i < j < n : a[i] - a[j] \rangle \uparrow \langle \uparrow i : 0 \leqslant i < n : a[i] - a[n] \rangle \\ = & \{ \ n \text{ not bounded } \} \\ & \langle \uparrow i, j : 0 \leqslant i < j < n : a[i] - a[j] \rangle \uparrow (\langle \uparrow i : 0 \leqslant i < n : a[i] \rangle - a[n]). \end{array}$$

So we strengthen the invariant by adding a variable s satisfying

$$Q \equiv s = \langle \uparrow i : 0 \leqslant i < n : a[i] \rangle.$$

The invariant is $P \wedge Q \wedge 0 \leqslant n \leqslant N$.

The program:

$$\begin{array}{ll} \textbf{con } N : Int \ \{0 \leqslant N\} & \\ \textbf{con } a : \textbf{array } [0..N) \textbf{ of } Int & \\ \textbf{var } r, s, n : Int & \\ r, s, n := -\infty, -\infty, 0 & \text{-- Pf0} \\ \{P \wedge Q \wedge 0 \leqslant n \leqslant N, bnd : N - n\} & \text{-- Pf1} \\ \textbf{do } n \neq N \rightarrow & \\ \quad r, s, n := r \uparrow (s - a[n]), s \uparrow a[n], n + 1 & \text{-- Pf2} \\ \textbf{od} & \\ \{r = \langle \uparrow i \ j : 0 \leqslant i < j < N : a[i] - a[j] \rangle\} & \text{-- Pf3} \end{array}$$

Here I am omitting other proofs and presenting only Pf2:

Pf2

$$\begin{array}{ll} & (P \wedge Q \wedge 0 \leqslant n \leqslant N)[r, s, n \backslash r \uparrow (s - a[n]), \ s \uparrow a[n], \ n + 1] \\ \equiv & r \uparrow (s - a[n]) = \langle \uparrow i, j : 0 \leqslant i < j < n + 1 : a[i] - a[j] \rangle \wedge \\ & s \uparrow a[n] = \langle \uparrow i : 0 \leqslant i < n + 1 : a[i] \rangle \ \wedge \ 0 \leqslant n + 1 \leqslant N \\ \Leftarrow & \{ \text{ split off } i = n \ \} \\ & r \uparrow (s - a[n]) = \langle \uparrow i, j : 0 \leqslant i < j < n : a[i] - a[j] \rangle \uparrow (\langle \uparrow i : 0 \leqslant i < n : a[i] \rangle - a[n]) \wedge \\ & s \uparrow a[n] = \langle \uparrow i : 0 \leqslant i < n : a[i] \rangle + a[n] \ \wedge \ 0 \leqslant n < N \\ \Leftarrow & r = \langle \uparrow i, j : 0 \leqslant i < j < n : a[i] - a[j] \rangle \wedge \\ & s = \langle \uparrow i : 0 \leqslant i < n : a[i] \rangle \ \wedge \ 0 \leqslant n \leqslant N \ \wedge n \neq N \\ \equiv & P \wedge Q \wedge 0 \leqslant n \leqslant N \wedge n \neq N. \end{array}$$

4. Derive a solution for:

   $\mathbf{con}\ N : Int\{N \geqslant 1\};\ a : \mathbf{array}\ [0..N)\ \mathbf{of}\ Int$
   $\mathbf{var}\ r : Int$
   S
   $\{r = \langle \#i, j : 0 \leqslant i < j < N : a[i] \times a[j] \geqslant 0 \rangle\}$.

Solution: Replace constant N by variable n, and use a loop that increments n in each step. Let

$$P \equiv r = \langle \#i, j : 0 \leqslant i < j < n : a[i] \times a[j] \geqslant 0 \rangle.$$

We first attempt to use $P \wedge 0 \leqslant n \leqslant N$ as the invariant and, apparently, $N - n$ as the bound. To find out how to update r such that we may increment n, we calculate (assuming $0 \leqslant n$):

$$\begin{array}{ll} & \langle \#i, j : 0 \leqslant i < j < n + 1 : a[i] \times a[j] \geqslant 0 \rangle \\ = & \{ \text{ since } 0 \leqslant n, \text{ splitting off } j = n \ \} \\ & \langle \#i, j : 0 \leqslant i < j < n : a[i] \times a[j] \geqslant 0 \rangle + \langle \#i : 0 \leqslant i < n : a[i] \times a[n] \geqslant 0 \rangle. \end{array}$$

To further simplify $\langle \#i : 0 \leqslant i < n : a[i] \times a[n] \geqslant 0 \rangle$, we do a case analysis on $a[n]$:

$$\langle \#i : 0 \leqslant i < n : a[i] \times a[n] \geqslant 0 \rangle = \begin{array}{ll} \langle \#i : 0 \leqslant i < n : a[i] \geqslant 0 \rangle, & \text{if } a[n] > 0; \\ n, & \text{if } a[n] = 0; \\ \langle \#i : 0 \leqslant i < n : a[i] \leqslant 0 \rangle, & \text{if } a[n] < 0. \end{array}$$

Thus we strengthen the invariant by adding two more variables:

$$Q_1 \equiv s_1 = \langle \#i : 0 \leqslant i < n : a[i] \geqslant 0 \rangle \ ,$$
$$Q_2 \equiv s_2 = \langle \#i : 0 \leqslant i < n : a[i] \leqslant 0 \rangle \ .$$

The invariant is $P \wedge Q_1 \wedge Q_2 \wedge 0 \leqslant n \leqslant N$.

The program:

$\mathbf{con}\ N : Int\{N \geqslant 1\};\ a : \mathbf{array}\ [0..N)\ \mathbf{of}\ Int$
$\mathbf{var}\ r, s_1, s_2, n : Int$

$r, s_1, s_2, n := 0, 0, 0, 0$
$\{P \wedge Q_1 \wedge Q_2 \wedge 0 \leqslant n \leqslant N,\ \ bnd : N - n\}$
$\mathbf{do}\ n < N \rightarrow$
  $\mathbf{if}\ a[n] > 0 \rightarrow r,\ s_1,\ n := r + s_1,\ s_1 + 1,\ n + 1$
  $\mid\ a[n] = 0 \rightarrow r,\ s_1,\ s_2,\ n := r + n,\ s_1 + 1,\ s_2 + 1,\ n + 1$
  $\mid\ a[n] < 0 \rightarrow r,\ s_2,\ n := r + s_2,\ s_2 + 1,\ n + 1$
  $\mathbf{fi}$
$\mathbf{od}$
$\{r = \langle \#i\, j : 0 \leqslant i < j < N : a[i] \times a[j] \geqslant 0 \rangle\}$.

---

5. Consider again the maximum segment sum problem and its derivation in the handouts. Since the computation of r requires the value of the subterm $\langle \uparrow p : 0 \leqslant p \leqslant n + 1 : sum\ p\ (n + 1) \rangle$, some would think it makes more sense to use the following loop invariant $P_0 \wedge P_1 \wedge Q$, where

   $P_0 \equiv r = \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : sum\ p\ q \rangle$,
   $P_1 \equiv s = \langle \uparrow p : 0 \leqslant p \leqslant n + 1 : sum\ p\ (n + 1) \rangle$,
   $Q \equiv 0 \leqslant n \leqslant N$.

What situation could you run into, if you try to construct a program using the invariant above? What if the array is non-empty, that is, $1 \leqslant N$?

Solution: With $0 \leqslant N$ we would run into problem initialising s. The initialisation may look like:

$$r, s, n := 0, ?, 0$$
$$\{P_0 \wedge P_1 \wedge Q\}$$

and the value of s should be

$$\begin{aligned}
& P_1[r, s, n \backslash 0, ?, 0] \\
\equiv\ & ? = \langle \uparrow p : 0 \leqslant p \leqslant 1 : \mathit{sum}\ p\ 1 \rangle \\
\equiv\ & ? = \mathit{sum}\ 0\ 1 + \mathit{sum}\ 1\ 1 \\
\equiv\ & ? = f[0] + 0 \\
\equiv\ & ? = f[0] \ .
\end{aligned}$$

However, $f[0]$ does not have a value when $N = 0$.

When we have $1 \leqslant N$ instead of $0 \leqslant N$ we will be able to initialize the variables by $r, s, n := 0, f[0], 0$. When we construct the loop body, knowing that $P_0[n \backslash n + 1] \equiv r = r \uparrow \langle \uparrow p : 0 \leqslant p \leqslant (n + 1) : \mathit{sum}\ p\ (n + 1) \rangle$, the constructed loop body would be:

$$\begin{aligned}
& \{P_0 \wedge P_1 \wedge 0 \leqslant n \leqslant N \wedge n \neq N\} \\
& r := r \uparrow s \\
& \{P_0[n \backslash n + 1] \wedge P_1 \wedge 0 \leqslant n + 1 \leqslant N\} \\
& s := (s + f[n + 1]) \uparrow 0 \\
& \{(P_0 \wedge P_1 \wedge 0 \leqslant n \leqslant N)[n \backslash n + 1]\} \\
& n := n + 1 \\
& \{P_0 \wedge P_1 \wedge 0 \leqslant n \leqslant N\}
\end{aligned}$$

Note that the order of assignments is different from that in the handouts.

However, upon termination (that is, $n = N$) we would need to establish

$$s = \langle \uparrow p : 0 \leqslant p \leqslant N + 1 : \mathit{sum}\ p\ (N + 1) \rangle \ ,$$

which we cannot do because $f[N]$ is not defined.

It is possible to fix all these: for example, terminate the loop one step earlier and do some post processing, and put the entire loop under an **if** to ensure that $1 \leqslant N$. The resulting program would not be as clean as the one in the handouts, though.