Programming Languages:
Imperative Program Construction
6. Loop Construction II: Strengthening the Invariant

Shin-Cheng Mu

Autumn. 2021

1 Maximum Segment Sum

A classical problem: given an array of integers, find largest possible sum of a consecutive segment.

$\mathbf{con}\ N : \mathit{Int}\ \{0 \leqslant N\}$
$\mathbf{con}\ f : \mathbf{array}\ [0..N)\ \mathbf{of}\ \mathit{Int}$
S
$\{r = \langle\uparrow p\ q : 0 \leqslant p \leqslant q \leqslant N : \mathit{sum}\ p\ q\rangle\}$

where $\mathit{sum}\ p\ q = \langle\Sigma i : p \leqslant i < q : f[i]\rangle$.

Details That Matter

- Note the use of $\leqslant$ and $<$ in the specification.
- The range in $\mathit{sum}\ p\ q$ is $p \leqslant i < q$. It computes the sum of $f\ [p..q)$ — not including $f[q]$!
- Therefore when $p = q$, $\mathit{sum}\ p\ q$ computes the sum of an empty segment.
- In the postcondition we have $p \leqslant q$ — we allow empty segments in our solution!
- We must have $q \leqslant N$ instead of $q < N$. Otherwise segments containing the rightmost element would not be valid solutions.

Previously Introduced Techniques

- Replace N by n. Use $P \wedge Q$ as the invariant, where

  $$P \equiv r = \langle\uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : \mathit{sum}\ p\ q\rangle\ ,$$
  $$Q \equiv 0 \leqslant n \leqslant N\ .$$

- Use $\neg\ (n = N)$ as guard. This way we immediately have that $P \wedge Q \wedge n = N$ imply the desired postcondition.
- How do we know we want $0 \leqslant n \leqslant N$? It can be forced by our development later. But let's expedite the pace.
- Initialisation: $n, r := 0, 0$.
- Use $N - n$ as the bound.
- To decrease the bound, let $n := n + 1$ be the last statement of the loop.

We get this program.

$\mathbf{con}\ N : \mathit{Int}\ \{0 \leqslant N\}$
$\mathbf{con}\ f : \mathbf{array}\ [0..N)\ \mathbf{of}\ \mathit{Int}$
$\mathbf{var}\ r, n : \mathit{Int}$
$r, n := 0, 0$
$\{P \wedge Q, \mathit{bnd} : N - n\}$
$\mathbf{do}\ n \neq N \rightarrow\ ???\ ; n := n + 1\ \mathbf{od}$
$\{r = \langle\uparrow p\ q : 0 \leqslant p \leqslant q \leqslant N : \mathit{sum}\ p\ q\rangle\}$

Now we need to construct the $???$ part.

Constructing the Loop Body

How to construct the $???$ part?

$\{P \wedge Q \wedge n \neq N\}$
$???$
$\{(P \wedge Q)[n \backslash n + 1]\}$
$n := n + 1$
$\{P \wedge Q\}$

Constructing Assignments

How do you construct such an assignment?

$\{r = \langle\uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : \mathit{sum}\ p\ q\rangle \wedge$
$\quad Q \wedge n \neq N\}$
$r :=\ ???$
$\{(P \wedge Q)[n \backslash n + 1]\}$
$n := n + 1$
$\{P \wedge Q\}$

Recall what we have learnt: if from $(P \wedge Q)[n\backslash n+1]$ we can infer that

$$r = \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : sum\ p\ q \rangle \oplus E \ \ ,$$

the statement $???$ could be $r := r \oplus E$.

Examining the Expression

To reason about $P[n\backslash n+1]$, we calculate (assuming $P \wedge Q \wedge n \neq N$):

$$\begin{aligned}
& \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : sum\ p\ q \rangle [n\backslash n+1] \\
= & \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n+1 : sum\ p\ q \rangle \\
= & \quad \{ \text{split off } q = n+1 \text{, see next slide} \} \\
& \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : sum\ p\ q \rangle \uparrow \\
& \quad \langle \uparrow p : 0 \leqslant p \leqslant (n+1) : sum\ p\ (n+1) \rangle \\
= & \quad \{ P_0 \} \\
& r \uparrow \langle \uparrow p : 0 \leqslant p \leqslant (n+1) : sum\ p\ (n+1) \rangle \ \ .
\end{aligned}$$

Therefore we wish to update r by:

$$r := r \uparrow \langle \uparrow p : 0 \leqslant p \leqslant (n+1) : sum\ p\ (n+1) \rangle \ \ .$$

But $\langle \uparrow p : 0 \leqslant p \leqslant (n+1) : sum\ p\ (n+1) \rangle$ cannot be computed in one step!

We could compute $\langle \uparrow p : 0 \leqslant p \leqslant (n+1) : sum\ p\ (n+1) \rangle$ in a loop…or can we store it in another variable?

Splitting Off?

Regarding the step "split off $q = n+1$":

$$\begin{aligned}
& 0 \leqslant p \leqslant q \leqslant n+1 \\
= & 0 \leqslant p \leqslant q \wedge q \leqslant n+1 \\
= & 0 \leqslant p \leqslant q \wedge (q \leqslant n \vee q = n+1) \\
= & (0 \leqslant p \leqslant q \wedge q \leqslant n) \vee (0 \leqslant p \leqslant q \wedge q = n+1) \\
= & 0 \leqslant p \leqslant q \leqslant n \vee (0 \leqslant p \leqslant q \wedge q = n+1) \ \ .
\end{aligned}$$

Note that for the second step to be valid, we need $-1 \leqslant n$ (which is implied by $0 \leqslant n \leqslant N$). Always remember to check that the range is non-empty before you split!

Therefore we have (abbreviating $sum…$ to R):

$$\begin{aligned}
& \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n+1 : R \rangle \\
= & \quad \{ \text{previous calculation} \} \\
& \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n \vee (0 \leqslant p \leqslant q \wedge q = n+1) : R \rangle \\
= & \quad \{ \text{range split (8.16)} \} \\
& \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : R \rangle \uparrow \\
& \quad \langle \uparrow p\ q : 0 \leqslant p \leqslant q \wedge q = n+1 : R \rangle \\
= & \quad \{ \text{one-point rule} \} \\
& \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : R \rangle \uparrow \\
& \quad \langle \uparrow p : 0 \leqslant p \leqslant n+1 : R \rangle \ \ .
\end{aligned}$$

Things to note:

- Calculation for other patterns of ranges (e.g. $0 \leqslant p \leqslant q \leqslant n+1$) are slightly different. Watch out!
- In practice, the "splitting off" step is but one quick step. We do not do the reasoning above in such detail.
- We show you the details above for expository purpose.
- In other problems we may see slightly different ranges, such as $0 \leqslant p < q < n+1$. The result of splitting is different too. Take extra care!

Strengthening the Invariant

Knowing that we need to update r with $\langle \uparrow p : 0 \leqslant p \leqslant (n+1) : sum\ p\ (n+1) \rangle$, let us store it in some variable! Introduce a new variable s, and *strengthen* the invariant to $P_0 \wedge P_1 \wedge Q$, where

$$\begin{aligned}
P_0 &\equiv r = \langle \uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : sum\ p\ q \rangle \ \ , \\
P_1 &\equiv s = \langle \uparrow p : 0 \leqslant p \leqslant n : sum\ p\ n \rangle \ \ , \\
Q\ &\equiv 0 \leqslant n \leqslant N \ \ .
\end{aligned}$$

Maximum Suffix Sum

- That is, while r is the maximum *segment* sum so far, s is the maximum *suffix* sum so far.
- We discover the need of this concept through symbolic calculation.
- This is a pattern for many "segment problems": *to solve a problem about segments, solve a suffix problem for all prefixes.*

Q: Why don't we let $s = \langle \uparrow p : 0 \leqslant p \leqslant n+1 : sum\ p\ (n+1) \rangle$?

A: For this example you will run into some problems. The details are left as an exercise. But in general it is not always a bad idea.

Constructing the Loop Body

Therefore, a possible strategy would be:

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

Updating the Prefix Sum

Recall $P_1 \equiv s = \langle\uparrow p : 0 \leqslant p \leqslant n : sum\ p\ n\rangle$.

$$\begin{aligned}
&\langle\uparrow p : 0 \leqslant p \leqslant n : sum\ p\ n\rangle[n\backslash n+1] \\
=\ &\langle\uparrow p : 0 \leqslant p \leqslant n+1 : sum\ p\ (n+1)\rangle \\
=\ &\quad \{\text{ splitting off } p = n+1 \} \\
&\langle\uparrow p : 0 \leqslant p \leqslant n : sum\ p\ (n+1)\rangle \uparrow \\
&\quad sum\ (n+1)\ (n+1) \\
=\ &\quad \{\ [n+1..n+1) \text{ is an empty range } \} \\
&\langle\uparrow p : 0 \leqslant p \leqslant n : sum\ p\ (n+1)\rangle \uparrow 0 \\
=\ &\quad \{\text{ splitting off } i = n \text{ in } sum \} \\
&\langle\uparrow p : 0 \leqslant p \leqslant n : sum\ p\ n + f[n]\rangle) \uparrow 0 \\
=\ &\quad \{\text{ distributivity } \} \\
&(\langle\uparrow p : 0 \leqslant p \leqslant n : sum\ p\ n\rangle + f[n]) \uparrow 0 \enspace .
\end{aligned}$$

Thus, $\{P_1\}\ s :=\ ?\ \{P_1[n\backslash n+1]\}$ is satisfied by $s := (s + f[n]) \uparrow 0$.

A Key Property

- The last step labelled "distributivity" uses a rule mentioned before: provided that $\neg occurs(i, F)$ and R non-empty:

$$F + \langle\uparrow i : R : S\rangle = \langle\uparrow i : R : F + S\rangle$$
$$F + \langle\downarrow i : R : S\rangle = \langle\downarrow i : R : F + S\rangle \enspace .$$

- The rules are valid because addition distributes into maximum/minimum:

$$x + (y \uparrow z) = (x + y) \uparrow (x + z) \enspace ,$$
$$x + (y \downarrow z) = (x + y) \downarrow (x + z) \enspace .$$

- That is the key property that allows us to have an efficient algorithm for the maximum segment sum problem!
- Through calculation, we not only have an algorithm, but also identified the key property that makes it work, which we can generalise to other problems.

Derived Program

$$\begin{aligned}
&\mathbf{con}\ N : Int\ \{0 \leqslant N\} \\
&\mathbf{con}\ f : \mathbf{array}\ [0..N)\ \mathbf{of}\ Int \\
&\mathbf{var}\ r, s, n : Int \\
&r, s, n := 0, 0, 0 \\
&\{P_0 \wedge P_1 \wedge Q, bnd : N - n\} \\
&\mathbf{do}\ n \neq N \rightarrow \\
&\quad s := (s + f[n]) \uparrow 0 \\
&\quad r := r \uparrow s \\
&\quad n := n + 1 \\
&\mathbf{od} \\
&\{r = \langle\uparrow p\ q : 0 \leqslant p \leqslant q \leqslant N : sum\ p\ q\rangle\}
\end{aligned}$$

$$\begin{aligned}
P_0 &\equiv r = \langle\uparrow p\ q : 0 \leqslant p \leqslant q \leqslant n : sum\ p\ q\rangle) \enspace , \\
P_1 &\equiv s = \langle\uparrow p : 0 \leqslant p \leqslant n : sum\ p\ n\rangle) \enspace , \\
Q &\equiv 0 \leqslant n \leqslant N \enspace .
\end{aligned}$$

"Strengthening"?

- We stay that the invariant $P_0 \wedge P_1 \wedge Q$ is "stronger" than $P \wedge Q$ because the former promises more.
- The resulting loop computes values for two variables rather than one.
- However, the program ends up being quicker because more results from the previous iteration of the loop can be utilised.
- It is a common phenomena: a generalised theorem is easier to prove.
- We will see another way to generalise the invariant in the rest of the course.

Lessons Learnt?

Let the symbols do the work!

- We discover how to strengthen the invariant by calculating and finding out what is missing.
- Expressions are your friend, and blind guessing can be minimised. We always get some clue from the expressions.
- Since we rely only on the symbols, the same calculation/algorithm can be generalised to other problems (e.g. as long as the same distributivity propery holds).

If we remove the pre/postconditions and the invariant, can you tell us what the program does?

- Without the assertions, programs mean nothing. The assertions are what matter about the program.
- Structured programming is not about making (the operational parts of) code easier to read/understand.
- Such efforts are bound to end in vain: even a simple three-line loop can be hard to understand if the assertions, encoding the intentions of the programmer, are stripped away.
- Instead, structured programming is about organising the code around the structure of the proofs.

- Once the pre/postconditions are given, and the invariants and bounds are determined, one can derive the code accordingly.

- It is pointless arguing, for example, "using a *break* here makes the code easier to read."

- One shall not need to "understand" the operational parts of the code, but to check whether it meets the specification.

2 No. of Pairs in an Array

Consider constructing the following program:

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

Previously Introduced Techniques

- Replace N by n. Use $P \wedge Q$ as the invariant, where

$$\begin{aligned} P \equiv\ & r = \langle \# i, j : 0 \leqslant i < j < n : \\ & \qquad a[i] \leqslant 0 \wedge a[j] \geqslant 0 \rangle, \\ Q \equiv\ & 0 \leqslant n \leqslant N. \end{aligned}$$

- Use $\neg\ (n = N)$ as guard. This way we immediately have that $P \wedge Q \wedge n = N$ imply the desired postcondition.

- Initialisation: $n, r := 0, 0$.

- Use $N - n$ as the bound.

- To decrease the bound, let $n := n + 1$ be the last statement of the loop.

We get this program.

$$\begin{array}{l}\textbf{con } N : \mathit{Int}\ \{0 \leqslant N\};\ a : \textbf{array } [0..N) \textbf{ of } \mathit{Int}\\ \textbf{var } r, n : \mathit{Int}\\ r, n := 0, 0\\ \{P \wedge Q, \mathit{bnd} : N - n\}\\ \textbf{do } n \neq N \rightarrow ...; n := n + 1 \textbf{ od}\\ \{r = \langle \# i\ j : 0 \leqslant i < j < N : a[i] \leqslant 0 \wedge a[j] \geqslant 0\rangle\}\end{array}$$

Now we need to construct the ... part.

Constructing the Loop Body

How to construct the ... part?

$$\begin{array}{l}\{P \wedge Q \wedge n \neq N\}\\ \ldots\\ \{(P \wedge Q)[n \backslash n + 1]\}\\ n := n + 1\\ \{P \wedge Q\}\end{array}$$

No. of Pairs in an Array

To reason about $P[n \backslash n + 1]$, we calculate (assuming $P \wedge Q \wedge n \neq N$):

$$\begin{array}{cl} & \langle \# i, j : 0 \leqslant i < j < n + 1 : a[i] \leqslant 0 \wedge a[j] \geqslant 0 \rangle \\ = & \{ \text{ split off } j = n \text{, see the next slide } \} \\ & \langle \# i, j : 0 \leqslant i < j < n : a[i] \leqslant 0 \wedge a[j] \geqslant 0 \rangle + \\ & \langle \# i : 0 \leqslant i < n : a[i] \leqslant 0 \wedge a[n] \geqslant 0 \rangle \\ = & \{ \ P \ \} \\ & r + \langle \# i : 0 \leqslant i < n : a[i] \leqslant 0 \wedge a[n] \geqslant 0 \rangle \\ = & \begin{cases} r, & \text{if } a[n] < 0; \\ r + \langle \# i : 0 \leqslant i < n : a[i] \leqslant 0 \rangle, & \text{if } a[n] \geqslant 0. \end{cases} \end{array}$$

Let us try storing $\langle \# i : 0 \leqslant i < n : a[i] \leqslant 0 \rangle$ in another variable?

Splitting Off?

For expository purpose let us exam how the splitting was done:

$$\begin{aligned} & 0 \leqslant i < j < n + 1 \\ = {} & 0 \leqslant i < j \wedge j < n + 1 \\ = {} & 0 \leqslant i < j \wedge (j < n \vee j = n) \\ = {} & (0 \leqslant i < j \wedge j < n) \vee (0 \leqslant i < j \wedge j = n) \\ = {} & 0 \leqslant i < j < n \vee (0 \leqslant i < j \wedge j = n) \ . \end{aligned}$$

The second step is valid if $0 \leqslant n$.

A Frequent Pattern

We may see this pattern often. For some $\star$, we need to calculate:

$$\begin{array}{cl} & \langle \star i\ j : 0 \leqslant i < j < n + 1 : R \rangle \\ = & \{ \text{ previous calculation } \} \\ & \langle \star i\ j : 0 \leqslant i < j < n \vee (0 \leqslant i < j \wedge j = n) : R \rangle \\ = & \langle \star i\ j : 0 \leqslant i < j < n : R \rangle \star \\ & \quad \langle \star i\ j : 0 \leqslant i < j \wedge j = n : R \rangle \\ = & \{ \text{ one-point rule } \} \\ & \langle \star i\ j : 0 \leqslant i < j < n : R \rangle \star \\ & \quad \langle \star i : 0 \leqslant i < n : R \rangle \ . \end{array}$$

Calculation for other ranges (e.g. $0 \leqslant i \leqslant j \leqslant n+1$) are slightly different. Watch out!

Strengthening the Invariant

New plan: define

$$\begin{aligned} P_0 &\equiv\ r = \langle\, \#i, j : 0 \leqslant i < j < n : \\ &\qquad\qquad a[i] \leqslant 0 \wedge a[j] \geqslant 0\,\rangle, \\ P_1 &\equiv\ s = \langle\, \#i : 0 \leqslant i < n : a[i] \leqslant 0\,\rangle, \\ Q &\equiv\ 0 \leqslant n \leqslant N, \end{aligned}$$

and try to derive

$$\begin{array}{l} \mathbf{con}\ N : \mathit{Int}\ \{N \geqslant 0\};\ a : \mathbf{array}\ [0..N)\ \mathbf{of}\ \mathit{Int} \\ \mathbf{var}\ n, r, s : \mathit{Int} \\ \\ n, r, s := 0, 0, 0 \\ \{P_0\ \wedge\ P_1\ \wedge\ Q, \mathit{bnd} : N - n\} \\ \mathbf{do}\ n \neq N \rightarrow \ldots n := n + 1\ \mathbf{od} \\ \{r = \langle\, \#i, j : 0 \leqslant i < j < N : a[i] \leqslant 0 \wedge a[j] \geqslant 0\,\rangle\} \end{array}$$

Update the New Variable

$$\begin{array}{ll} & \langle\, \#i : 0 \leqslant i < n : a[i] \leqslant 0\,\rangle[n\backslash n+1] \\ = & \langle\, \#i : 0 \leqslant i < n+1 : a[i] \leqslant 0\,\rangle \\ = & \{\ \text{split off } i = n \text{ (assuming } 0 \leqslant n\text{)}\ \} \\ & \langle\, \#i : 0 \leqslant i < n : a[i] \leqslant 0\,\rangle + \#(a[n] \leqslant 0) \\ = & \{\ P_1\ \} \\ & s + \#(a[n] \leqslant 0) \\ = & \begin{cases} s & \text{if } a[n] > 0, \\ s+1 & \text{if } a[n] \leqslant 0. \end{cases} \end{array}$$

Resulting Program

$$\begin{array}{l} \ldots \{N \geqslant 0\} \\ n, r, s := 0, 0, 0 \\ \{P_0 \wedge P_1 \wedge Q, \mathit{bnd} : N - n\} \\ \mathbf{do}\ n \neq N \rightarrow \{P_0 \wedge P_1 \wedge Q \wedge n \neq N\} \\ \quad \mathbf{if}\ \ a[n] < 0 \rightarrow \mathit{skip} \\ \quad\ \ \mid a[n] \geqslant 0 \rightarrow r := r + s \\ \quad \mathbf{fi} \\ \quad \{P_0[n\backslash n+1] \wedge P_1 \wedge Q \wedge n \neq N\} \\ \quad \mathbf{if}\ \ a[n] > 0 \rightarrow \mathit{skip} \\ \quad\ \ \mid a[n] \leqslant 0 \rightarrow s := s + 1 \\ \quad \mathbf{fi} \\ \quad \{(P_0 \wedge P_1 \wedge Q)[n\backslash n+1]\} \\ \quad n := n + 1 \\ \mathbf{od} \\ \{r = \langle\, \#i, j : 0 \leqslant i < j < N : a[i] \leqslant 0 \wedge a[j] \geqslant 0\,\rangle\} \end{array}$$

Resulting Program

Since $P_0 \wedge P_1 \wedge Q \wedge n \neq N$ is a common precondition for the **if**'s (the second **if** does not use P_0), they can be combined:

$$\begin{array}{l} \ldots \{N \geqslant 0\} \\ n, r, s := 0, 0, 0 \\ \{P_0 \wedge P_1 \wedge Q, \mathit{bnd} : N - n\} \\ \mathbf{do}\ n \neq N \rightarrow \{P_0 \wedge P_1 \wedge Q \wedge n \neq N\} \\ \quad \mathbf{if}\ \ a[n] < 0 \rightarrow s := s + 1 \\ \quad\ \ \mid\ a[n] = 0 \rightarrow r, s := r + s, s + 1 \\ \quad\ \ \mid\ a[n] > 0 \rightarrow r := r + s \\ \quad \mathbf{fi} \\ \quad \{(P_0 \wedge P_1 \wedge Q)[n\backslash n+1]\} \\ \quad n := n + 1 \\ \mathbf{od} \\ \{r = \langle\, \#i, j : 0 \leqslant i < j < N : a[i] \leqslant 0 \wedge a[j] \geqslant 0\,\rangle\} \end{array}$$

Isn't It Getting A Bit Too Complicated?

- Quantifier and indexes manipulation tend to get very long and tedious.
  - Expect to see even longer expressions later!
- To certain extent, it is a restriction of the data structure we are using. With arrays we have to manipulate the indexes.
- Is it possible to use higher-level data structures? Lists? Trees?
  - Heap-allocated data structure with pointers is a horrifying beast!
  - Trying to be more abstract lead to further developments in programming languages, e.g. algebraic datatypes.