Programming Languages: Imperative Program Construction 5. Loop Construction I

Shin-Cheng Mu

Autumn. 2024

Correct by Construction

> Dijkstra: "The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness. But one should not first make the program and then prove its correctness, because then the requirement of providing the proof would only increase the poor programmer's burden. On the contrary: the programmer should ..."
>
> "...[let] correctness proof and program grow hand in hand: with the choice of the structure of the correctness proof one designs a program for which this proof is applicable."

Deriving Programs from Specifications

- From such a specification:

$$\begin{array}{l}\textbf{con}\ declarations\\ \{preconditions\}\\ prog\\ \{postcondition\}\end{array}$$

- we hope to derive $prog$.
- We usually work backwards from the post condition.
- The techniques we are about to learn is mostly about constructing loops and loop invariants.

1 Taking Conjuncts as Invariants

Conjunctive Postconditions

- When the post condition has the form $P \wedge Q$, one may take one of the conjuncts as the invariant and the other as the guard:
  - $\{P\}\ \textbf{do}\ \neg Q \to S\ \textbf{od}\ \{P \wedge Q\}$.
- In some extreme cases, since $P \equiv true \wedge P$, one may try:
  - $\{True\}\ \textbf{do}\ \neg P \to S\ \textbf{od}\ \{P\}$.
- E.g. to sort four variables:

$$\begin{array}{l}\{True\}\\ \textbf{do}\ a > b \to a, b := b, a\\ \ \ |\ \ \ b > c \to b, c := c, b\\ \ \ |\ \ \ c > d \to c, d := d, c\\ \textbf{od}\\ \{a \leqslant b \leqslant c \leqslant d\}\end{array}$$

- *Why does it terminate?*

Integral Division and Reminder

- Consider the specficication:

$$\begin{array}{l}\textbf{con}\ A, B : Int\{0 \leqslant A \wedge 0 < B\}\\ \textbf{var}\ q, r : Int\\ divmod\\ \{q = A\ \textbf{div}\ B \wedge r = A\ \textbf{mod}\ B\}\ .\end{array}$$

- The post condition expands to $A = q \times B + r \wedge 0 \leqslant r \wedge r < B$.

But Which Conjunct to Choose?

- $q = A\ \textbf{div}\ B \wedge r = A\ \textbf{mod}\ B$ expands to $A = q \times B + r \wedge 0 \leqslant r \wedge r < B$. Denote it by R. It leads to a number of possibilities:
- $\{0 \leqslant r \wedge r < B\}\ \textbf{do}\ A \neq q \times B + r \to S\ \textbf{od}\ \{R\}$,
- $\{A = q \times B + r \wedge r < B\}\ \textbf{do}\ 0 > r \to S\ \textbf{od}\ \{R\}$, or
- $\{A = q \times B + r \wedge 0 \leqslant r\}\ \textbf{do}\ r \geqslant B \to S\ \textbf{od}\ \{R\}$, etc.

Computing the Quotient and the Remainder

Try $A = q \times B + r \wedge 0 \leqslant r$ as the invariant and $\neg(r < B)$ as the guard:

$$
\begin{array}{l}
q, r := 0, A \\
\{P : A = q \times B + r \wedge 0 \leqslant r\} \\
\textbf{do } B \leqslant r \rightarrow \{P \wedge B \leqslant r\} \\
\qquad\qquad\quad q := q + 1 \\
\qquad\qquad\quad \{P'\} \\
\qquad\qquad\quad r := r - B \\
\qquad\qquad\quad \{P\} \\
\textbf{od} \\
\{P \wedge r < B\}
\end{array}
$$

- P is established by $q, r := 0, A$.
- Choose r as the bound.
- Since $B > 0$, try $r := r - B$:

$$
\begin{array}{ll}
& P[r \backslash r - B] \\
\equiv & A = q \times B + r - B \ \wedge \ 0 \leqslant r - B \\
\equiv & A = (q - 1) \times B + r \ \wedge \ B \leqslant r.
\end{array}
$$

  Hmm... we almost have $P \ \wedge \ B \leqslant r$, apart from that q is replaced by $q - 1$. Denote it by P'.

- $$
\begin{array}{ll}
& P'[q \backslash q + 1] \\
\equiv & A = (q + 1 - 1) \times B + r \ \wedge \ B \leqslant r \\
\equiv & A = q \times B + r \ \wedge \ B \leqslant r.
\end{array}
$$

  Thus we know that we want the assignment $q := q + 1$.

2 On Constructing Assignments

Updating a Variable

We will see this pattern often:

- We want to establish:

$$
\begin{array}{l}
\{x = E \ \wedge \ \ldots\} \\
x := x \oplus E' \\
\{x = E \oplus E'\}
\end{array}
$$

- It works because:

$$
\begin{array}{ll}
& (x = E \oplus E')[x \backslash x \oplus E'] \\
\equiv & x \oplus E' = E \oplus E' \\
\Leftarrow & x = E.
\end{array}
$$

- In general, given a function f, to establish:

$$
\begin{array}{l}
\{x = E\} \\
x := \ldots \\
\{x = f \ E\}
\end{array}
$$

- we can use an assignment $x{:=}f \ x$. It works because

$$
\begin{array}{ll}
& (x = f \ E)[x \backslash f \ x] \\
\equiv & f \ x = f \ E \\
\Leftarrow & x = E \ .
\end{array}
$$

3 Replacing Constants by Variables

Exponentiation

- Consider the problem:

$$
\begin{array}{l}
\textbf{con } A, B : \mathit{Int} \ \{A \geqslant 0 \ \wedge \ B \geqslant 0\} \\
\textbf{var } r : \mathit{Int} \\
\mathit{exponentiation} \\
\{r = A^B\} \ .
\end{array}
$$

- There is not much we can do with a state space consisting of only one variable.
- Replacing constants by variables may yield some possible invariants.
- Again we have several choices: $r = x^B$, $r = A^x$, $r = x^y$, etc.

Exponentiation

- Use the invariant $P_0 : r = A^x$, thus $P_0 \ \wedge \ x = B$ implies the post-condition.
- Strategy: increment x in the loop. An upper bound $P_1 : x \leqslant B$.
- $(r = A^x)[x \backslash x + 1] \equiv r = A^{x+1}$. However, when $r = A^x$ holds, $A^{x+1} = A \times A^x = A \times r$!
- Indeed,

$$
\begin{array}{ll}
& (r = A^{x+1})[r \backslash A \times r] \\
\equiv & A \times r = A^{x+1} \\
\Leftarrow & r = A^x.
\end{array}
$$

$r, x := 1, 0$
$\{r = A^x \ \wedge \ x \leqslant B, bnd : B - x\}$
do $x \neq B \rightarrow$
$r := A \times r$
$\{r = A^{x+1} \ \wedge \ x + 1 \leqslant B\}$
$x := x + 1$
od
$\{r = A^B\}$

Summing Up an Array

- Another simple exercise.
- We talk about it because we need range splitting.

con $N : Int \ \{0 \leqslant N\}$; $f : \textbf{array} \ [0..N) \ \textbf{of} \ Int$
var $x : Int$
sum
$\{x = \langle \Sigma i : 0 \leqslant i < N : f[i] \rangle\}$

Summing Up an Array

con $N : Int \ \{0 \leqslant N\}$; $f : \textbf{array} \ [0..N) \ \textbf{of} \ Int$;

$n, x := 0, 0$
$\{P : x = \langle \Sigma i : 0 \leqslant i < n : f[i] \rangle \ \wedge \ 0 \leqslant n,$
$\quad bnd : N - n\}$
do $n \neq N \rightarrow \{P \wedge n \neq N\}$
$x := x + f[n]; \ \ n := n + 1 \ \{P\}$ **od**
$\{x = \langle \Sigma i : 0 \leqslant i < N : f[i] \rangle\}$

- Inv. is established by $n, x := 0, 0$.
- Use $N - n$ as bound, try incrementing n:

$$
\begin{array}{ll}
 & (x = \langle \Sigma i : 0 \leqslant i < n : f[i] \rangle \ \wedge \ 0 \leqslant n)[n \backslash n + 1] \\
\equiv & x = \langle \Sigma i : 0 \leqslant i < n + 1 : f[i] \rangle \ \wedge \ 0 \leqslant n + 1 \\
\Leftarrow & x = \langle \Sigma i : 0 \leqslant i < n + 1 : f[i] \rangle \ \wedge \ 0 \leqslant n \\
\equiv & x = \langle \Sigma i : 0 \leqslant i < n : f[i] \rangle + f[n] \ \wedge \ 0 \leqslant n.
\end{array}
$$

-

$$
\begin{array}{ll}
 & (x = \langle \Sigma i : 0 \leqslant i < n : f[i] \rangle + f[n] \ \wedge \ 0 \leqslant n) \\
 & \quad [x \backslash x + f[n]] \\
\equiv & x + f[n] = \langle \Sigma i : 0 \leqslant i < n : f[i] \rangle + f[n] \ \wedge \ 0 \leqslant n \\
\Leftarrow & x = \langle \Sigma i : 0 \leqslant i < n : f[i] \rangle \ \wedge \ 0 \leqslant n.
\end{array}
$$