Programming Languages:
Imperative Program Construction
8. Case Studies

Shin-Cheng Mu

Autumn Term, 2022

National Taiwan University and Academia Sinica

Faster Division

- Recall the problem:

$$\begin{array}{l} \textbf{con}\ A, B : Int\ \{0 \leqslant A \wedge 0 < B\} \\ \textbf{var}\ q, r : Int \\ \ ? \\ \{A = q \times B + r \wedge 0 \leqslant r < B\} \quad . \end{array}$$

- Recall: recognising the postcondition as a conjunction, we use $A = q \times B + r \wedge 0 \leqslant r$ as the invariant and $\neg\,(r < B)$ as the guard.

- The program we came up with:

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

- In each iteration of the loop, r is decreased by B.
- We can probably get a quicker program by decreasing r by ... $2 \times B$, when possible.
- What about decreasing r by $4 \times B$, $8 \times B$,... etc?

Strategy...

$$\textbf{con } A, B : Int \ \{0 \leqslant A \wedge 0 < B\}$$
$$\textbf{var } q, r, b, k : Int$$
$$\ldots$$
$$\{0 \leqslant k \wedge b = 2^k \times B \wedge A < b\}$$
$$\ldots$$
$$\{A = q \times b + r \wedge 0 \leqslant r < b \wedge$$
$$\quad 0 \leqslant k \wedge b = 2^k \times B, bnd : b\}$$
$$\textbf{do } b \neq B \rightarrow \ldots \textbf{od}$$
$$\{A = q \times B + r \wedge 0 \leqslant r < B\}$$

$$
\begin{array}{l}
\textbf{con } A, B : Int \ \{0 \leqslant A \wedge 0 < B\} \\
\textbf{var } q, r, b, k : Int \\
b, k := B, 0 \\
\textbf{do } b \leqslant A \rightarrow b, k := b \times 2, k + 1 \textbf{ od} \\
\{0 \leqslant k \wedge b = 2^k \times B \wedge A < b\} \\
q, r := 0, A \\
\{A = q \times b + r \wedge 0 \leqslant r < b \wedge \\
\quad 0 \leqslant k \wedge b = 2^k \times B, bnd : b\} \\
\textbf{do } b \neq B \rightarrow \\
\quad \textbf{if } r < b \,/\, 2 \rightarrow q, b, k := q \times 2, b \,/\, 2, k - 1 \\
\quad \mid b \,/\, 2 \leqslant r \rightarrow q, b, k, r := q \times 2 + 1, b \,/\, 2, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad k - 1, r - b \,/\, 2 \\
\quad \textbf{fi} \\
\textbf{od} \\
\{A = q \times B + r \wedge 0 \leqslant r < B\}
\end{array}
$$

Kaldewaij presented the following alternative. Do you prefer this program?

$$
\begin{array}{l}
\textbf{con}\ A, B : Int\ \{0 \leqslant A \wedge 0 < B\} \\
\textbf{var}\ q, r, b, k : Int \\
b, k := B, 0 \\
\textbf{do}\ b \leqslant A \to b, k := b \times 2, k + 1\ \textbf{od} \\
q, r := 0, A \\
\textbf{do}\ b \neq B \to \\
\quad q, b, k := q \times 2, b \,/\, 2, k - 1 \\
\quad \textbf{if}\ r < b\ \to skip \\
\quad \ \mid b \leqslant r \to q, r := q + 1, r - b \\
\quad \textbf{fi} \\
\textbf{od} \\
\{A = q \times B + r \wedge 0 \leqslant r < B\}
\end{array}
$$

- The program has the advantage that we do not need to have $b \,/\, 2$ in the guards.
- Note what the first assignment establishes:

$$
\begin{aligned}
&\{A = q \times b + r \wedge 0 \leqslant r < b \wedge \\
&\quad 0 \leqslant k \wedge b = 2^k \times B \wedge b \neq B\} \\
&q, b, k := q \times 2, b \,/\, 2, k - 1 \\
&\{A = q \times b + r \wedge 0 \leqslant r < 2 \times b \wedge \\
&\quad 0 \leqslant k \wedge b = 2^k \times B\}
\end{aligned}
$$

Binary Search Revisited

- Given a sorted array of N numbers and a key, either locate the position where the key resides in the array, or report that the key does not present in the array, in $O(\log N)$ time.
- A possible spec:

$$
\begin{array}{l}
\textbf{con}\ N, K : \mathit{Int}\ \{0 < N\} \\
\textbf{con}\ F : \textbf{array}\ [0..N)\ \textbf{of}\ \mathit{Int}\ \{F\ \mathit{ascending}\} \\
\textbf{var}\ l, r : \mathit{Int} \\
\mathit{bsearch} \\
\{F[l] = K \vee \ldots\}\quad .
\end{array}
$$

The van Gasteren-Feijen Approach

- Van Gasteren and Feijen pointed a surprising fact: binary search does not apply only to sorted lists!
- In fact, they believe that comparing binary search to searching for a word in a dictionary is a major educational blunder.
- Their binary search: let Φ be a predicate on two integers with some additional constraints to be given later:

$$
\begin{array}{l}
\mathbf{con}\ M, N : \mathit{Int}\ \{M < N \wedge \Phi\ M\ N \wedge \ldots\} \\
\mathbf{var}\ l, r : \mathit{Int} \\
\mathit{bsearch} \\
\{M \leqslant l < N \wedge \Phi\ l\ (l+1)\}\quad .
\end{array}
$$

- Invariant: $\Phi\ l\ r \wedge M \leqslant l < r \leqslant N$, loop guard: $l + 1 \neq r$.
- Initialisation: $l, r := M, N$.
- Bound: $r - l$.
- For any m such that $l < m < r$, we have $r - m < r - l$ and $m - l < r - l$. Therefore both $l := m$ and $r := m$ decrease the bound.

- For $l := m$ we calculate.

$$
\begin{aligned}
& (\Phi\ l\ r \wedge M \leqslant l < r \leqslant N)[l \backslash m] \\
\equiv\ & \Phi\ l\ m \wedge M \leqslant m < r \leqslant N \\
\Leftarrow\ & \Phi\ l\ m \wedge M \leqslant l < m < r \leqslant N \quad .
\end{aligned}
$$

- That $l < m < r$ is our assumption. The leftover $\Phi\ l\ m$ gives rise to a guarded command: $\Phi\ l\ m \rightarrow l := m$.
- The case with $r := m$ is similar.

$$
\begin{array}{l}
\{M < N \wedge \Phi\ M\ N\} \\
l, r := M, N \\
\{\Phi\ l\ r \wedge M \leqslant l < r \leqslant N, bnd : r - l\} \\
\textbf{do}\ l + 1 \neq r \rightarrow \\
\quad \{\ldots \wedge l + 2 \leqslant r\} \\
\quad m := \text{anything s.t. } l < m < r \\
\quad \{\ldots \wedge l < m < r\} \\
\quad \textbf{if}\ \Phi\ m\ r \rightarrow l := m \\
\quad \mid \Phi\ l\ m \rightarrow r := m \\
\quad \textbf{fi} \\
\textbf{od} \\
\{M \leqslant l < N \wedge \Phi\ l\ (l + 1)\}
\end{array}
$$

Note: $m := (l + r)\ /\ 2$ is a valid choice, thanks to the precondition that $l + 2 \leqslant r$.

Constraints on Φ

- But we need the **if** to be total.
- Therefore we demand a constrant on Φ:

$$\Phi\ l\ r \Rightarrow \Phi\ l\ m \vee \Phi\ m\ r, \ \text{if}\ l < m < r. \tag{1}$$

- Some Φ satisfying (1) (for F of appropriate type):
  - $\Phi\ l\ r \equiv F[l] \neq F[r]$,
  - $\Phi\ l\ r \equiv F[l] < F[r]$,
  - $\Phi\ l\ r \equiv F[l] \leqslant A \wedge A \leqslant F[r]$,
  - $\Phi\ l\ r \equiv F[l] \times F[r] \leqslant 0$,
  - $\Phi\ l\ r \equiv F[l] \vee F[r]$,
  - $\Phi\ l\ r \equiv \neg\,(Q\ l) \wedge Q\ r$.
- Van Gasteren and Feijen believe that $\Phi\ l\ r = F[l] \neq F[r]$ is a

Searching for a Key

- The case $\Phi\ l\ r \equiv \neg\ (Q\ l) \wedge Q\ r$ is worth special attention.
- Choose $Q\ i \equiv K < F[i]$ for some K.
- Therefore $\Phi\ l\ r \equiv F[l] \leqslant K < F[r]$.
- That constitutes the binary search we wanted!
- The postcondition: $M \leqslant l < N \wedge F[l] \leqslant K < F[l+1]$.
- Note that we do *not* yet need F to be sorted!
- The algorithm gives you some l such that $F[l] \leqslant K < F[l+1]$. If there are more than one such l, one is returned non-deterministically.

- That F is sorted comes in when we need to establish that there is at most one l satisfying the postcondition.
- That is, either $F[l] = K$, or $\neg \langle \exists i : M \leqslant i < N : F[i] = K \rangle$.

The Program… Or A Part Of It

- Let $\Phi\ l\ r = F[l] \leqslant K < F[r]$.
- Processing the array fragment $F\,[a\,..\,b]$:

$$
\begin{array}{l}
l, r := a, b \\
\{\Phi\ l\ r \wedge a \leqslant l < r \leqslant b, bnd : r - l\} \\
\textbf{do}\ l + 1 \neq r \rightarrow \\
\quad m := (l + r)\ /\ 2 \\
\quad \textbf{if}\ F[m] \leqslant K \rightarrow l := m \\
\quad\ \mid K < F[m] \rightarrow r := m \\
\quad \textbf{fi} \\
\textbf{od} \\
\{a \leqslant l < b \wedge F[l] \leqslant K < F[l + 1]\}
\end{array}
$$

- Note that $F[a]$ and $F[b]$ are never accessed.
- This program is not yet complete….

- But wait.. to apply the algorithm to the entire array, we need the precondition $\Phi\ 0\ N$, that is $F[0] \leqslant K < F[N]$. Is that true? (We don't even have $F[N]$.)

- One can rule out cases when the precondition do not hold (and also deal with empty array). E.g.

$$
\begin{array}{l}
\textbf{if } 0 = N \to p := \mathit{False} \\
\ \mid 0 < N \to \\
\quad \textbf{if } K < F[0] \to p := \mathit{False} \\
\quad\ \mid F[N-1] = K \to p, l := \mathit{True}, N-1 \\
\quad\ \mid F[0] \leqslant K < F[N-1] \to \\
\qquad\quad a, b := 0, N-1 \\
\qquad\quad \mathit{program\ above} \\
\qquad\quad p := F[l] = K \\
\quad \textbf{fi}
\end{array}
$$

- But there is a better way... introduce two pseudo elements!
- Let $F[-1] = -\infty$ and $F[N] = \infty$.
- Initially, $\Phi\ 0\ N$ is satisfied.
- In the code, $F[-1]$ and $F[N]$ are never accessed. Therefore we do not actually have to represent them!
- We need to be careful interpreting the result, once the main loop terminates, however.

Let $\Phi\ l\ r = F[l] \leqslant K < F[r]$.

$\textbf{con}\ N, K : Int\ \{0 \leqslant N\}$
$\textbf{con}\ F : \textbf{array}\ [0..N)\ \textbf{of}\ Int\ \{F\ ascending\ \wedge$
$\quad F[-1] = -\infty \wedge F[N] = \infty\}$
$\textbf{var}\ l, m, r : Int$
$\textbf{var}\ p : Bool$

$l, r := -1, N$
$\{\Phi\ l\ r \wedge -1 \leqslant l < r \leqslant N, bnd : r - l\}$
$\textbf{do}\ l + 1 \neq r \rightarrow$
$\quad m := (l + r)\ /\ 2$
$\quad \textbf{if}\ F[m] \leqslant K \rightarrow l := m$
$\quad\ |\ K < F[m] \rightarrow r := m$
$\quad \textbf{fi}$
$\textbf{od}$
$\{-1 \leqslant l < N \wedge F[l] \leqslant K < F[l + 1]\}$

$$
\begin{array}{l}
\{-1 \leqslant l < N \wedge F[l] \leqslant K < F[l+1]\} \\
\textbf{if} -1 = l \rightarrow p := \mathit{False} \\
\ \mid 0 \leqslant l \quad\ \ \rightarrow p := F[l] = K \\
\textbf{fi} \\
\{p = \langle \exists i : 0 \leqslant i < N : F[i] = K\rangle \wedge \\
\quad p \Rightarrow F[l] = K\}
\end{array}
$$

- Kaldewaij derived an alternative program that introduces only $F[N] = \infty$ (but not $F[-1] = -\infty$), while requiring the array to be non-empty.
- The main loop is the same. It is only post-loop interpretation that is different.

- Recall that Bentley proposed using binary search as an exercise.

- Bentley's solution can be rephrased below:

$$
\begin{array}{l}
l, r, p := 0, N-1, \mathit{False} \\
\textbf{do } l \leqslant r \rightarrow \\
\quad m := (l + r) \;/\; 2 \\
\quad \textbf{if } F[m] < K \rightarrow l := m + 1 \\
\quad \;\mid F[m] = k \rightarrow p := \mathit{True}; \mathit{break} \\
\quad \;\mid K < F[m] \rightarrow r := m - 1 \\
\quad \textbf{fi} \\
\textbf{od}
\end{array}
$$

I'd like to derive it, but

- it is harder to formally deal with *break*.
    - Still, Bentley employed a semi-formal reasoning using a loop invariant to argue for the correctness of the program.
- To relate the test $F[m] < K$ to $l := m + 1$ we have to bring in the fact that F is sorted earlier.

Comparison

- The two programs do not solve exactly the same problem (e.g. when there are multiple Ks in F).
- Is the second program quicker because it assigns l and r to $m+1$ and $m-1$ rather than m?
    - $l := m+1$ because $F[m]$ is covered in another case;
    - $r := m-1$ because a range is represented differently.
- Is it quicker to perform an extra test to *return* early?
    - When K is not in F, the test is wasted.
    - Rolfe claimed that single comparison is quicker in average.
    - Knuth: single comparison needs $17.5 \lg N + 17$ instructions, double comparison needs $18 \lg N - 16$ instructions.