Programming Languages: Imperative Program Construction 10. Swaps in Arrays

Shin-Cheng Mu

Autumn Term, 2021

National Taiwan University and Academia Sinica

Swaps

- Extend the notion of function alteration to two entries.

$$\begin{aligned}(f\!:\!x,y\!\shortrightarrow\! e_1,e_2)\;z &= e_1 \quad, \text{if } z = x,\\ &= e_2 \quad, \text{if } z = y,\\ &= f\,z \quad, \text{otherwise.}\end{aligned}$$

- Given array $h\;[0..N)$ and integer expressions E and F, let $swap\;h\;E\;F$ be a primitive operation such that:

$$\begin{aligned}&wp\;(swap\;h\;E\;F)\;P = def\;(h[E]) \wedge def\;(h[F]) \wedge\\ &\quad P[h\backslash(h\!:\!E,F\!\shortrightarrow\! h[F],h[E])]\quad.\end{aligned}$$

- Intuitively, $swap\;h\;E\;F$ means “swapping the values of $h[E]$ and $h[F]$. (See the notes below, however.)

- *swap h E F* does not always literally "swaps the values." For example, it is *not* always the case that

$$\{h[E] = X\}\ \mathit{swap}\ h\ E\ F\ \{h[F] = X\} \quad .$$

- Consider $h[0] = 0 \wedge h[1] = 1$. This does not hold:

$$\{h[h[0]] = 0\}\ \mathit{swap}\ h\ (h[0])\ (h[1])\ \{h[h[1]] = 0\} \quad .$$

- In fact, after swapping we have $h[0] = 1 \wedge h[1] = 0$, and hence $h[h[1]] = 1$.

- However, when h does not occur free in E and F, we do have

$$
\begin{aligned}
&(\{\langle \forall i : i \neq E \wedge i \neq F : h[i] = H\ i \rangle\} \wedge \\
&\quad h[E] = X \wedge h[F] = Y) \\
&\mathit{swap}\ h\ E\ F \\
&(\{\langle \forall i : i \neq E \wedge i \neq F : h[i] = H\ i \rangle\} \wedge \\
&\quad h[E] = Y \wedge h[F] = X) \quad .
\end{aligned}
$$

- It is a convenient rule we use when reasoning about swapping.

- Kaldewaij defined *swap h E F* as an abbreviation of

$$|[\ \mathbf{var}\ r; r := h[E]; h[E] := h[F]; h[F] := r\ ]| \quad ,$$

- where r is a fresh name and $|[\ldots]|$ denotes a program block with local constants and variables. We have not used this feature so far.
- I do not think this definition is correct, however. The definition would not behave as we expect if F refers to $h[E]$.

The Dutch National Flag

- Let $RWB = \{R, W, B\}$ (standing respecively for red, white, and blue).

$$
\begin{array}{l}
\textbf{con } N : Int \; \{0 \leqslant N\} \\
\textbf{var } h : \textbf{array } [0..N) \textbf{ of } RWB \\
\textbf{var } r, w : Int \\
dutch\_national\_flag \\
\{0 \leqslant r \leqslant w \leqslant N \land \\
\quad \langle \forall i : 0 \leqslant i < r : h[i] = R \rangle \land \\
\quad \langle \forall i : r \leqslant i < w : h[i] = W \rangle \land \\
\quad \langle \forall i : w \leqslant i < N : h[i] = B \rangle \land \}
\end{array}
$$

- The program shall manipulate h only by swapping.
- Denote the postcondition by Q.

White

- The case for white is the easiest, since

$$P_0 \wedge P_1 \wedge h[w] = W \Rightarrow \\ \quad (P_0 \wedge P_1)[w \backslash w + 1] \quad .$$

- It is sufficient to let S_w be simply $w := w + 1$.

- We have

$$
\begin{array}{l}
\{P_r \wedge P_w \wedge P_b \wedge w < b \wedge h[w] = B\} \\
swap\ h\ w\ (b-1) \\
\{P_r \wedge P_w \wedge P_b \wedge w < b \wedge h[b-1] = B\} \\
b := b - 1 \\
\{P_r \wedge P_w \wedge P_b \wedge w \leqslant b\}
\end{array}
$$

- Thus we choose $swap\ h\ w\ (b-1); b := b-1$ as S_b.

- Precondition: $P_r \wedge P_w \wedge P_b \wedge w < b \wedge h[w] = R$.
- It appears that $swap\ h\ w\ r$ establishes $P[w \backslash w + 1]$. But we have to see what $h[r]$ is before we can increment r.
- P_w implies $r < w \Rightarrow h[r] = W$. Equivalently, we have $r = w \vee h[r] = W$.

- We have

$$
\begin{array}{l}
\{P_r \wedge P_w \wedge P_b \wedge r = w < b \wedge h[w] = R\} \\
\mathit{swap}\ h\ w\ r \\
\{P_r \wedge P_w \wedge P_b \wedge w < b \wedge h[r] = R\} \\
r, w := r + 1, w + 1 \\
\{P_r \wedge P_w \wedge P_b \wedge r = w \leqslant b\}
\end{array}
$$

Red: Case $h[r] = W$

- We have

$$
\begin{aligned}
&\{P_r \wedge P_w \wedge P_b \wedge w < b \wedge h[r] = W \wedge h[w] = R\}\\
&\mathit{swap}\ h\ w\ r\\
&\{P_r \wedge h[r] = R \wedge \langle \forall i : r+1 \leqslant i < w : h[i] = W \rangle \wedge\\
&\quad h[w] = W \wedge P_b \wedge w < b\}\\
&r, w := r+1, w+1\\
&\{P_r \wedge P_w \wedge P_b \wedge r = w \leqslant b\}
\end{aligned}
$$

- In both cases, $\mathit{swap}\ h\ w\ r; r, w := r+1, w+1$ is a valid choice.

$$
\begin{array}{l}
\textbf{con}\ K, N : \mathit{Int}\ \{0 \leqslant K < N\} \\
\textbf{var}\ h : \textbf{array}\ [0..N)\ \textbf{of}\ A
\end{array}
$$

- $\{\langle \forall i : 0 \leqslant i < N : h[i] = H[i] \rangle\}$

  rotation

  $\{\langle \forall i : 0 \leqslant i < N : h[(i + K)\ \textbf{mod}\ N] = H[i] \rangle\}$.

- To eliminate **mod**, the postcondition can be rewritten as:

$$
\begin{array}{l}
\langle \forall i : 0 \leqslant i < N - K : h[i + K] = H[i] \rangle \wedge \\
\quad \langle \forall i : N - K \leqslant i < N : h[i + K - N] = H[i] \rangle \quad .
\end{array}
$$

- Or, $h[K..N) = H[0..N - K) \wedge h[0..K) = H[N - K..N)$.

- For this problem we benefit from using more abstract notations.
- Segments of arrays can be denoted by variables. E.g. $X = H[0..N-K)$ and $Y = H[N-K..N)$.
- Concatenation of arrays are denoted by juxtaposition. E.g. $H[0..N) = XY$.
- Empty sequence is denoted by $[\,]$.
- Length of a sequence X is denoted by $l\ X$.

- Specification:

$$\{h = XY\}$$
$$\textit{rotation}$$
$$\{h = YX\}$$

- When $l\ X = l\ Y$ we can establish the postcondition easily — just swap the corresponding elements.
- Denote swapping of equal-lengthed array segments by $SWAP\ X\ Y$.

- When $l\ X < l\ Y$, h can be written as $h = XUV$,
- where $l\ U = l\ X$ and $UV = Y$.
- Task:

$$\{h = XUV \wedge l\ U = l\ X\}$$
$$\mathit{rotation}$$
$$\{h = UVX\}$$

- Strategy:

$$\begin{aligned}
&\{h = XUV \wedge l\ U = l\ X\}\\
&SWAP\ X\ U\\
&\{h = UXV\}\\
&\ ??\\
&\{h = UVX\}
\end{aligned}$$

- The part ?? shall transform XV into VX — a problem having the same form as the original!
- Some (including myself) would then go for a recursive program. But there is another possibility.

- Consider the symmetric case where $l\ X > l\ Y$.

$$\{h = UVY \wedge l\ V = l\ Y\}$$
$$SWAP\ V\ Y$$
$$\{h = UYV\}$$
$$??$$
$$\{h = YUV\}$$

- In general, the array is of them form $AUVB$, where UV needs to be transformed into VU, while A and B are parts that are done.

- Strategy:

$$\begin{array}{l}
\{h = XY\} \\
A, U, V, B := [\,], X, Y, [\,] \\
\{h = AUVB \wedge YX = AVUB, bnd : l\ U + l\ V\} \\
\mathbf{do}\ U \neq [\,] \wedge V \neq [\,] \rightarrow \ldots \mathbf{od} \\
\{h = YX\}
\end{array}$$

- Call the invariant P. Intuitively it means "currently the array is $AUVB$, and if we exchange U and V, we are done."
- Note the choice of guard: $P \wedge (U = [\,] \wedge V = [\,]) \Rightarrow h = YX$.

$$
\begin{array}{l}
A, U, V, B := [\,], X, Y, [\,] \\
\{h = AUVB \wedge YX = AVUB, bnd : l\ U + l\ V\} \\
\textbf{do}\ U \neq [\,] \wedge V \neq [\,] \rightarrow \\
\quad \textbf{if}\ l\ U \geqslant l\ V \rightarrow \qquad \text{-- } l\ U_1 = l\ V \\
\qquad \{h = AU_0U_1VB \wedge YX = AVU_0U_1B\} \\
\qquad SWAP\ U_1\ V \\
\qquad \{h = AU_0VU_1B \wedge YX = AVU_0U_1B\} \\
\qquad U, B := U_0, U_1B \\
\qquad \{h = AUVB \wedge YX = AVUB\} \\
\quad \mid\ l\ U \leqslant l\ V \rightarrow \qquad \text{-- } l\ V_0 = l\ U \\
\qquad \{h = AUV_0V_1B \wedge YX = AV_0V_1UB\} \\
\qquad SWAP\ U\ V_0 \\
\qquad \{h = AV_0UV_1B \wedge YX = AV_0V_1UB\} \\
\qquad A, V := AV_0, V_1 \\
\qquad \{h = AUVB \wedge YX = AVUB\} \\
\quad \textbf{fi}
\end{array}
$$

- Introduce $a, b, k, l : Int$.
- $A = h[0..a)$;
- $U = h[a..a + k)$, hence $l\ U = k$;
- $V = h[b - l..n)$, hence $l\ V = l$;
- $B = h[b..N)$.
- Additional invariant: $a + k = b - l$.
- Why having both k and l? We will see later.

- Represented using indices:

$$
\begin{array}{l}
a, k, l, b := 0, N - K, K, N \\
\textbf{do}\ k \neq 0 \wedge l \neq 0 \rightarrow \\
\quad \textbf{if}\ k \geqslant l \rightarrow SWAP\ (b - l)\ l\ (-l) \\
\qquad\qquad\qquad k, b := k - l, b - l \\
\quad \mid\ k \leqslant l \rightarrow SWAP\ a\ k\ k \\
\qquad\qquad\qquad a, l := a + k, l - k \\
\quad \textbf{fi} \\
\textbf{od}
\end{array}
$$

- where $SWAP\ x\ num\ off$ abbreviates

$$
\begin{array}{l}
|[\ \textbf{var}\ n : Int \\
\quad n := x \\
\quad \textbf{do}\ n \neq x + num \rightarrow swap\ h\ n\ (n + off) \\
\qquad\qquad\qquad\qquad n := n + 1 \\
\quad \textbf{od}
\end{array}
$$

- To find out the number of swaps performed, we use a variable t to record the number of swaps.
- If we keep only computation related to t, k, and l:

$$
\begin{array}{l}
k, l, t := N - K, K, 0 \\
\textbf{do } k \neq 0 \wedge l \neq 0 \rightarrow \\
\quad \textbf{if } k \geqslant l \rightarrow t := t + l; k := k - l \\
\quad \mid k \leqslant l \rightarrow t := t + k; l := l - k \\
\quad \textbf{fi} \\
\textbf{od}
\end{array}
$$

- Observe: the part concerning k and l resembles computation of greatest common divisor.
- In fact, $gcd\ k\ l = gcd\ N\ (N - K)$, which is $gcd\ N\ K$.
- When the program terminates, $k + l = gcd\ N\ K$.
- It's always true that $t + k + l = N$.
- Therefore, the total number of swaps is $t = N - (k + l) = N - gcd\ N\ K$.