Programming Languages: Imperative Program Construction 10. Swaps in Arrays

Shin-Cheng Mu

Autumn Term, 2021

National Taiwan University and Academia Sinica

Swaps

Swaps

- Extend the notion of function alteration to two entries.

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

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

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

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

Complications

- *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 $h[h[1]] = 1$.

A Simpler Case

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

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

- It is a convenient rule we use when reasoning about swapping.
- Note that, in the rule above, E and F are expressions, while X, Y, H are logical variables.

Note: Kaldewaij's Swap

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

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

- where r is a fresh name and $|[...]|$ 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

The Dutch National Flag

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

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

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

White

- The case for white is the easiest, since

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

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

- We have

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

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

Red

- 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 $r = w \vee h[r] = W$.

Red: Case $r = w$

- We have

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

Red: Case $h[r] = W$

- We have

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

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

con $K, N : Int\ \{0 \leqslant K < N\}$
var h : **array** $[0..N)$ **of** A

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

rotation

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

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

$$\langle \forall i : 0 \leqslant i < N - K : h[i + K] = H[i] \rangle \wedge \\ \langle \forall i : N - K \leqslant i < N : h[i + K - N] = H[i] \rangle \ .$$

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

Abstract Notations

- 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:

  $\{h = XY\}$
  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*.

Thinking Lengths

- 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:

$\{h = XUV \wedge l\ U = l\ X\}$
$SWAP\ X\ U$
$\{h = UXV\}$
??
$\{h = UVX\}$

- 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.

Leading to an Invariant...

- 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.

The Invariant

- Strategy:

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

- 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: $P \wedge (U = [\,] \wedge V = [\,]) \Rightarrow h = YX$.

An Abstract Program

$A, U, V, B := [\,], X, Y, [\,]$

$\{h = AUVB \wedge YX = AVUB, bnd : l\ U + l\ V\}$

do $U \neq [\,] \wedge V \neq [\,] \rightarrow$

  if $l\ U \geqslant l\ V \rightarrow$ -- $l\ U_1 = l\ V$

    $\{h = AU_0U_1VB \wedge YX = AVU_0U_1B\}$

    $SWAP\ U_1\ V$

    $\{h = AU_0VU_1B \wedge YX = AVU_0U_1B\}$

    $U, B := U_0, U_1B$

    $\{h = AUVB \wedge YX = AVUB\}$

  | $l\ U \leqslant l\ V \rightarrow$ -- $l\ V_0 = l\ U$

    $\{h = AUV_0V_1B \wedge YX = AV_0V_1UB\}$

    $SWAP\ U\ V_0$

    $\{h = AV_0UV_1B \wedge YX = AV_0V_1UB\}$

    $A, V := AV_0, V_1$

    $\{h = AUVB \wedge YX = AVUB\}$

  fi

Representing the Sequences

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

A Concrete Program

- Represented using indices:

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

- where *SWAP x num off* abbreviates

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

Greatest Common Divisor

- 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 $t = N - (k + l) = N - gcd\ N\ K$.