Programming Languages: Imperative Program Construction 9. Array Manipulation

Shin-Cheng Mu

Autumn Term, 2022

Materials in these notes are mainly from Kaldewaij [Kal90]. Some examples are adapted from the course CSci 550: Program Semantics and Derivation taught by Prof. H. Conrad Cunningham [Cun06], University of Mississippi.

1 Some Notes on Definedness

Assignment Revisited

- Recall the weakest precondition for assignments:

$$wp\ (x := E)\ P = P[x \backslash E] \quad .$$

- That is not the whole story... since we have to be sure that E is defined!

Definedness

- In our current language, given expression E there is a systematic (inductive) definition on what needs to be proved to ensure that E is defined. Let's denote it by $def\ E$.
- We will not go into the detail but give examples.
- For example, if there is division in E, the denominator must not be zero.
  - $def\ (x + y\ /\ (z + x)) = (z + x \neq 0)$.
  - $def\ (x + y\ /\ 2) = (2 \neq 0) = True$.

Weakest Precondition

- A more complete rule:

$$wp\ (x := E)\ P = P[x \backslash E] \land def\ E \quad .$$

- In fact, all expressions need to be defined. E.g.

$$\begin{array}{l} wp\ (\mathbf{if}\ B_0 \to S_0 \mid B_1 \to S_1\ \mathbf{fi})\ P = \\ \quad B_0 \Rightarrow wp\ S_0\ P \land B_1 \Rightarrow wp\ S_1\ P \land (B_0 \lor B_1) \land \\ \quad def\ B_0 \land def\ B_1 \quad . \end{array}$$

How come we have never mentioned so?

- How come we have never mentioned so?
- The first partial operation we have used was division. And the denominator was usually a constant (namely, 2!).

Array Bound

- Array indexing is a partial operation too — we need to be sure that the index is within the domain of the array.
- Let $A : \mathbf{array}\ [M..N)\ \mathbf{of}\ Int$ and let I be an expression. We define $def\ (A[I]) = def\ I \land M \leqslant I < N$.
- E.g. given $A : \mathbf{array}\ [0..N)\ \mathbf{of}\ Int$,
  - $def\ (A[x\ /\ z] + A[y]) = z \neq 0 \land 0 \leqslant x\ /\ z < N \land 0 \leqslant y < N$.
  - $wp\ (s := s \uparrow A[n])\ P = P[s \backslash s \uparrow A[n]] \land 0 \leqslant n < N$.
- We never made it explicit, because conditions such as $0 \leqslant n < N$ were usually already in the invariant/guard and thus discharged immediately.

2 Array Assignment

- So far, all our arrays have been constants — we read from the arrays but never wrote to them!

- Consider a : **array** $[0..2)$ **of** Int, where $a[0] = 1$ and $a[1] = 1$.
- It should be true that

$$\begin{array}{l}\{a[0] = 1 \land a[1] = 1\} \\ a[a[1]] := 0 \\ \{a[a[1]] = 1\} \quad .\end{array}$$

- However, if we use the previous wp,

$$\begin{array}{ll} & wp\ (a[a[1]] := 0)\ (a[a[1]] = 1) \\ \equiv & (a[a[1]] = 1)[a[a[1]] \backslash 0] \\ \equiv & 0 = 1 \\ \equiv & False \quad . \end{array}$$

- What went wrong?

Another Counterexample

- For a more obvious example where our previous wp does not work for array assignment:
- $wp\ (a[i] := 0)\ (a[2] \neq 0)$ appears to be $a[2] \neq 0$, since $a[i]$ does not appear (verbatim) in $a[2] \neq 0$.
- But what if $i = 2$?

Arrays as Functions

- An array is a function. E.g. a : **array** $[0..N)$ **of** $Bool$ is a function $Int \rightarrow Bool$ whose domain is $[0..N)$.
- Indexing $a[n]$ is function application.
  - Some textbooks use the same notation for function application and array indexing.
  - (Could that have been a better choice for this course?)

Function Alteration

- Given $f : A \rightarrow B$, let $(f : x \mapsto e)$ denote the function that *maps* x *to* e, and otherwise the same as f.

$$\begin{array}{lll}(f : x \mapsto e)\ y & = e & \text{, if } x = y; \\ & = f\ y & \text{, otherwise.}\end{array}$$

- For example, given $f\ x = x^2$, $(f : 1 \mapsto -1)$ is a function such that

$$\begin{array}{ll}(f : 1 \mapsto -1)\ 1 = -1 & , \\ (f : 1 \mapsto -1)\ x = x^2 & \text{, if } x \neq 1.\end{array}$$

wp for Array Assignment

- Key: assignment to array should be understood as altering the entire function.
- Given a : **array** $[M..N)$ **of** A (for any type A), the updated rule:

$$\begin{array}{rl} wp\ (a[I] := E)\ P = & P[a \backslash (a : I \mapsto E)] \land \\ & def\ (a[I]) \land def\ E \quad . \end{array}$$

- In our examples, $def\ (a[I])$ and $def\ E$ can often be discharged immediately. For example, the boundary check $M \leqslant I < N$ can often be discharged soon. But do not forget about them.

The Example

- Recall our example

$$\begin{array}{l}\{a[0] = 1 \land a[1] = 1\} \\ a[a[1]] := 0 \\ \{a[a[1]] = 1\} \quad .\end{array}$$

- We aim to prove

$$\begin{array}{l} a[0] = 1 \land a[1] = 1 \Rightarrow \\ \quad wp\ (a[a[1]] := 0)\ (a[a[1]] = 1) \quad . \end{array}$$

Assume $a[0] = 1 \land a[1] = 1$.

$$\begin{array}{ll} & wp\ (a[a[1]] := 0)\ (a[a[1]] = 1) \\ \equiv & \{\ \text{def. of } wp \text{ for array assignment}\ \} \\ & (a : a[1] \mapsto 0)[(a : a[1] \mapsto 0)[1]] = 1 \\ \equiv & \{\ \text{assumption: } a[1] = 1\ \} \\ & (a : 1 \mapsto 0)[(a : 1 \mapsto 0)[1]] = 1 \\ \equiv & \{\ \text{def. of alteration: } (a : 1 \mapsto 0)[0] = 0\ \} \\ & (a : 1 \mapsto 0)[0] = 1 \\ \equiv & \{\ \text{def. of alteration: } (a : 1 \mapsto 0)[0] = a[0]\ \} \\ & a[0] = 1 \\ \equiv & \{\ \text{assumption: } a[0] = 1\ \} \\ & True \quad . \end{array}$$

Restrictions

- In this course, parallel assignments to arrays are not allowed.
- This is done to avoid having to define what the following program ought to do:

$$\begin{array}{l} x, y := 0, 0; \\ a[x], a[y] := 0, 1 \end{array}$$

- It is possible to give such programs a definition (e.g. choose an order), but we prefer to keep it simple.

3 Typical Array Manipulation in a Loop

3.1 All Zeros

Consider:

$$\begin{array}{l} \mathbf{con}\ N : \mathit{Int}\ \{0 \leqslant N\} \\ \mathbf{var}\ h : \mathbf{array}\ [0..N)\ \mathbf{of}\ \mathit{Int} \\ \mathit{allzeros} \\ \{\langle \forall i : 0 \leqslant i < N : h[i] = 0 \rangle\} \end{array}$$

The Usual Drill

$$\begin{array}{l} \mathbf{con}\ N : \mathit{Int}\ \{0 \leqslant N\} \\ \mathbf{var}\ h : \mathbf{array}\ [0..N)\ \mathbf{of}\ \mathit{Int} \\ \mathbf{var}\ n : \mathit{Int} \\ n := 0 \\ \{\langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle \wedge 0 \leqslant n \leqslant N, \\ \quad \mathit{bnd} : N - n\} \\ \mathbf{do}\ n \neq N \rightarrow\ ? \\ \qquad\qquad\qquad n := n + 1 \\ \mathbf{od} \\ \{\langle \forall i : 0 \leqslant i < N : h[i] = 0 \rangle\} \end{array}$$

Constructing the Loop Body

- With $0 \leqslant n \leqslant N \wedge n \neq N$:

$$\begin{array}{ll} & \langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle[n \backslash n+1] \\ \equiv & \langle \forall i : 0 \leqslant i < n+1 : h[i] = 0 \rangle \\ \equiv & \quad \{\ \text{split off } i = n\ \} \\ & \langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle \wedge h[n] = 0 \quad . \end{array}$$

- If we conjecture that $?$ is an assignment $h[I] := E$, we ought to find I and E such that the following can be satisfied:

$$\begin{array}{l} \langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle \wedge 0 \leqslant n < N \Rightarrow \\ \quad \langle \forall i : 0 \leqslant i < n : (h : I \mapsto E)[i] = 0 \rangle \wedge \\ \quad (h : I \mapsto E)[n] = 0 \quad . \end{array}$$

- An obvious choice: $(h : n \mapsto 0)$,
- which almost immediately leads to

$$\begin{array}{ll} & \langle \forall i : 0 \leqslant i < n : (h : n \mapsto 0)[i] = 0 \rangle \wedge \\ & (h : n \mapsto 0)[n] = 0 \\ \equiv & \quad \{\ \text{function alteration}\ \} \\ & \langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle \wedge 0 = 0 \\ \Leftarrow & \langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle \wedge 0 \leqslant n < N \quad . \end{array}$$

The Program

$$\begin{array}{l} \mathbf{con}\ N : \mathit{Int}\ \{0 \leqslant N\} \\ \mathbf{var}\ h : \mathbf{array}\ [0..N)\ \mathbf{of}\ \mathit{Int} \\ \mathbf{var}\ n : \mathit{Int} \\ n := 0 \\ \{\langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle \wedge 0 \leqslant n \leqslant N, \\ \quad \mathit{bnd} : N - n\} \\ \mathbf{do}\ n \neq N \rightarrow h[n] := 0; n := n + 1\ \mathbf{od} \\ \{\langle \forall i : 0 \leqslant i < N : h[i] = 0 \rangle\} \end{array}$$

Obvious, but useful.

3.2 Simple Array Assignment

- The calculation can certainly be generalised.
- Given a function $H : \mathit{Int} \rightarrow A$, and suppose we want to establish

  $$\langle \forall i : 0 \leqslant i < N : h[i] = H\ i \rangle \quad ,$$

  where H *does not depend on* h (e.g, h does not occur free in H).
- Let $P\ n = 0 \leqslant n < N \wedge \langle \forall i : 0 \leqslant i < n : h[i] = H\ i \rangle$).
- We aim to establish $P\ (n+1)$, given $P\ n \wedge n \neq N$.
- One can prove the following:

  $$\begin{array}{l} \{P\ n \wedge n \neq N \wedge E = H\ n\} \\ h[n] := E \\ \{P\ (n+1)\} \quad , \end{array}$$

- which can be used in a program fragment...

  $$\begin{array}{l} \{P\ 0\} \\ n := 0 \\ \{P\ n, \mathit{bnd} : N - n\} \\ \mathbf{do}\ n \neq N \rightarrow \\ \qquad \{\ \text{establish } E = H\ n\ \} \\ \quad h[n] := E \\ \quad n := n + 1 \\ \mathbf{od} \\ \{\langle \forall i : 0 \leqslant i < N : h[i] = H\ i \rangle\} \end{array}$$

- Why do we need E? Isn't E simply $H\ n$?
- In some cases $H\ n$ can be computed in one expression. In such cases we can simply do $h[n] := H\ n$.
- In some cases E may refer to previously computed results — other variables, or even h.
  - Yes, E may refer to h while H does not. There are such examples in the Practicals.

3.3 Histogram

Consider:

$$\begin{array}{l}\textbf{con } N : \mathit{Int}\ \{0 \leqslant N\}; X : \textbf{array } [0..N) \textbf{ of } \mathit{Int} \\ \{\langle \forall i : 0 \leqslant i < N : 1 \leqslant X[i] \leqslant 6\rangle\} \\ \textbf{var } h : \textbf{array } [1..6] \textbf{ of } \mathit{Int} \\ \mathit{histogram} \\ \{\langle \forall i : 1 \leqslant i \leqslant 6 : h[i] = \\ \quad \langle \# k : 0 \leqslant k < N : X[k] = i\rangle\rangle\}\end{array}$$

The Up Loop Again

- Let $P\ n$ denote $\langle \forall i : 1 \leqslant i \leqslant 6 : h[i] = \langle \# k : 0 \leqslant k < n : X[k] = i\rangle\rangle$.
- A program skeleton:

$$\begin{array}{l}\textbf{con } N : \mathit{Int}\ \{0 \leqslant N\}; X : \textbf{array } [0..N) \textbf{ of } \mathit{Int} \\ \{\langle \forall i : 0 \leqslant i < N : 1 \leqslant X[i] \leqslant 6\rangle\} \\ \textbf{var } h : \textbf{array } [1..6] \textbf{ of } \mathit{Int}; n : \mathit{Int} \\ \mathit{initialise} \\ n := 0 \\ \{P\ n \wedge 0 \leqslant n \leqslant N, \mathit{bnd} : N - n\} \\ \textbf{do } n \neq N \rightarrow\ ? \\ \qquad\qquad n := n + 1 \\ \textbf{od} \\ \{\langle \forall i : 1 \leqslant i \leqslant 6 : h[i] = \\ \quad \langle \# k : 0 \leqslant k < N : X[k] = i\rangle\rangle\}\end{array}$$

- The *initialise* fragment has to satisfy $P\ 0$, that is

$$\begin{array}{ll} & \langle \forall i : 1 \leqslant i \leqslant 6 : h[i] = \langle \# k : 0 \leqslant k < 0 : X[k] = i\rangle\rangle \\ \equiv & \langle \forall i : 1 \leqslant i \leqslant 6 : h[i] = 0\rangle\ ,\end{array}$$

- which can be performed by *allzeros*.

Constructing the Loop Body

- Let's calculate $P\ (n+1)$, assuming $0 \leqslant n < N$:

$$\begin{array}{ll} & \langle \forall i : 1 \leqslant i \leqslant 6 : h[i] = \\ & \quad \langle \# k : 0 \leqslant k < n + 1 : X[k] = i\rangle\rangle \\ \equiv & \quad \{\text{split off } k = n\} \\ & \langle \forall i : 1 \leqslant i \leqslant 6 : h[i] = \\ & \quad \langle \# k : 0 \leqslant k < n : X[k] = i\rangle + \#(X[n] = i)\rangle\end{array}$$

- Recall that $\# : \mathit{Bool} \rightarrow \mathit{Int}$ is the function such that

$$\begin{array}{l}\#\ \mathit{False} = 0 \\ \#\ \mathit{True}\ = 1\ .\end{array}$$

- Again we conjecture that $h[I] := E$ will do the trick.
- We want to find I ane E such that $P\ n \wedge 0 \leqslant n < N \Rightarrow (P\ (n+1))[h \backslash (h : I \mapsto E)]$ can be proved.
- Assume $P\ n \wedge 0 \leqslant n < N$, consider $(P\ (n+1))[h \backslash (h : I \mapsto E)]$

$$\begin{array}{ll} & \langle \forall i : 1 \leqslant i \leqslant 6 : (h : I \mapsto E)[i] = \\ & \quad \langle \# k : 0 \leqslant k < n : X[k] = i\rangle + \#(X[n] = i)\rangle \\ \equiv & \quad \{P\ n\} \\ & \langle \forall i : 1 \leqslant i \leqslant 6 : (h : I \mapsto E)[i] = \\ & \quad h[i] + \#(X[n] = i)\rangle \\ \equiv & \quad \{\text{defn. of } \#\} \\ & \langle \forall i : 1 \leqslant i \leqslant 6 : (h : I \mapsto E)[i] = V\ i\rangle, \textbf{where} \\ & \quad V\ i = h[i] + 1\ \text{, if } X[n] = i; \\ & \qquad\qquad h[i]\ \text{, if } X[n] \neq i. \\ \equiv & \quad \{\text{function alteration}\} \\ & \langle \forall i : 1 \leqslant i \leqslant 6 : (h : I \mapsto E)[i] = \\ & \quad (h : X[n] \mapsto h[i] + 1)[i]\rangle\ .\end{array}$$

- Therefore one chooses $I = X[n]$ and $E = h[X[n]] + 1$.

The Program

Let $P\ n \equiv \langle \forall i : 1 \leqslant i \leqslant 6 : h[i] = \langle \# k : 0 \leqslant k < n : X[k] = i\rangle\rangle$.

$$\begin{array}{l}\textbf{con } N : \mathit{Int}\ \{0 \leqslant N\}; X : \textbf{array } [0..N) \textbf{ of } \mathit{Int} \\ \{\langle \forall i : 0 \leqslant i < N : 1 \leqslant X[i] \leqslant 6\rangle\} \\ \textbf{var } h : \textbf{array } [1..6] \textbf{ of } \mathit{Int} \\ \textbf{var } n : \mathit{Int} \\ n := 1 \\ \textbf{do } n \neq 7 \rightarrow h[n] := 0; n := n + 1 \textbf{ od} \\ \{P\ 0\} \\ n := 0 \\ \{P\ n \wedge 0 \leqslant n \leqslant N, \mathit{bnd} : N - n\} \\ \textbf{do } n \neq N \rightarrow h[X[n]] := h[X[n]] + 1 \\ \qquad\qquad n := n + 1 \\ \textbf{od} \\ \{\langle \forall i : 1 \leqslant i \leqslant 6 : h[i] = \\ \quad \langle \# k : 0 \leqslant k < N : X[k] = i\rangle\rangle\}\end{array}$$

4 Swaps

- Extend the notion of function alteration to two entries.

$$\begin{array}{rll}(f : x, y \mapsto e1, e2)\ z & = e1 & \text{, if } z = x, \\ & = e2 & \text{, if } z = y, \\ & = f\ z & \text{, otherwise.}\end{array}$$

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

  $wp\ (swap\ h\ E\ F)\ P = def\ (h[E]) \wedge def\ (h[F]) \wedge$
  $\quad P[h \backslash (h : E, F \mapsto h[F], h[E])]\ \ .$

- 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

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

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

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

- 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

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

- 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 [Kal90, Chapter 10] 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 |[...]| 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 $h[E]$.

4.1 The Dutch National Flag

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

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

- The program shall manipulate h only by swapping.

- Denote the postcondition by Q.

Invariant

- Introduce a variable b.

- Choose as invariant $P_0 \wedge P_1$, where

  $$\begin{array}{ll}P_0 & \equiv P_r \wedge P_w \wedge P_b \\ P_1 & \equiv 0 \leqslant r \leqslant w \leqslant b \leqslant N \\ P_r & \equiv \langle\forall i : 0 \leqslant i < r :\ \ h[i] = R\rangle \\ P_w & \equiv \langle\forall i : r \leqslant i < w :\ h[i] = W\rangle \\ P_b & \equiv \langle\forall i : b \leqslant i < N : h[i] = B\rangle\end{array}$$

- $P_0 \wedge P_1$ can be established by $r, w, b := 0, 0, N$.

- If $w = b$, we get the postcondition Q.

The Plan

$$\begin{array}{l}r, w, b := 0, 0, N \\ \{P_0 \wedge P_1, bnd : b - w\} \\ \mathbf{do}\ b \neq w \rightarrow \mathbf{if}\ h[w] = R\ \ \rightarrow S_r \\ \qquad\qquad\quad\ \ |\ \ h[w] = W \rightarrow S_w \\ \qquad\qquad\quad\ \ |\ \ h[w] = B\ \ \rightarrow S_b \\ \qquad\qquad\quad\ \mathbf{fi} \\ \mathbf{od} \\ \{Q\}\end{array}$$

Observation

- Note that
  - r is the number of red elements detected,
  - $w - r$ is the number of white elements detected,

- $N-b$ is the number of blue elements detected.

- Therefore, S_w should contain $w := w+1$, S_b should contain $b := b-1$·
- S_r should contain $r, w := r+1, w+1$, thus r increases but $w-r$ is unchanged.
- The bound decreases in all cases! Good sign.

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 $w := w+1$.

Blue

- We have

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

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

Red

- Precondition: $P_r \wedge P_w \wedge P_b \wedge w < b \wedge h[w] = R$.
- It appears that $\mathit{swap}\ h\ w\ r$ 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

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

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

The Program

$$\begin{array}{l}
\textbf{con}\ N : \mathit{Int}\ \{0 \leqslant N\} \\
\textbf{var}\ h : \textbf{array}\ [0..N)\ \textbf{of}\ \mathit{RWB} \\
\textbf{var}\ r, w, b : \mathit{Int} \\
r, w, b := 0, 0, N \\
\{P_0 \wedge P_1, \mathit{bnd} : b - w\} \\
\textbf{do}\ b \neq w \rightarrow \textbf{if}\ h[w] = R \ \rightarrow \mathit{swap}\ h\ w\ r \\
\qquad\qquad\qquad\qquad\qquad\quad r, w := r+1, w+1 \\
\qquad\qquad\quad \mid\ h[w] = W \rightarrow w := w+1 \\
\qquad\qquad\quad \mid\ h[w] = B \ \rightarrow \mathit{swap}\ h\ w\ (b-1) \\
\qquad\qquad\qquad\qquad\qquad\quad b := b-1 \\
\qquad\qquad\quad \textbf{fi} \\
\textbf{od} \\
\{Q\}
\end{array}$$

4.2 Rotation

Rotation

- Given: $h : \textbf{array}\ [0..N)\ \textbf{of}\ A$ with integer constants $0 \leqslant K < N$.
- Task: rotate h over K places. That is, $h[0]$ is moved to $h[K]$, $h[1]$ to $h[(1+K)\ \textbf{mod}\ N]$, $h[2]$ to $h[(2+K)\ \textbf{mod}\ N]$...
- using swap operations only.

Specification

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

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

- To eliminate $\mathbf{mod}$, 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:

$$\begin{array}{l} \{h = XY\} \\ rotation \\ \{h = YX\} \end{array}$$

- 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:

$$\begin{array}{l} \{h = XUV \wedge l\ U = l\ X\} \\ rotation \\ \{h = UVX\} \end{array}$$

- Strategy:

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

- 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 $l\ X > l\ Y$.

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

- 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:

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

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

An Abstract Program

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

Representing the Sequences

- Introduce $a, b, k, l : Int$.
- $A = h[0..a)$;
- $U = h[a..a + k)$, 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:

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

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

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

- that is, starting from index x, swap num elements with those off positions away.

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:

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

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

References

[Cun06] H. C. Cunningham. CSci 550: Program Semantics and Derivation. University of Mississippi. https://john.cs.olemiss.edu/~hcc/csci550/, 2006.

[Kal90] A. Kaldewaij. *Programming: the Derivation of Algorithms*. Prentice Hall, 1990.