Programming Languages: Imperative Program Construction 9. Array Manipulation

Shin-Cheng Mu

Autumn Term, 2022

National Taiwan University and Academia Sinica

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

Some Notes on Definedness

- 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!

- 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 $\mathit{def}\ E$.
- We will not go into the detail but give examples.
- For example, if there is division in E, the denominator must not be zero.
  - $\mathit{def}\ (x + y \ / \ (z + x)) = (z + x \neq 0)$.
  - $\mathit{def}\ (x + y \ / \ 2) = (2 \neq 0) = \mathit{True}$.

- A more complete rule:

$$wp\ (x := E)\ P = P[x \backslash E] \wedge \mathit{def}\ E \quad .$$

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

$$\begin{aligned} &wp\ (\mathbf{if}\ B_0 \rightarrow S_0 \mid B_1 \rightarrow S_1\ \mathbf{fi})\ P = \\ &\quad B_0 \Rightarrow wp\ S_0\ P \wedge B_1 \Rightarrow wp\ S_1\ P \wedge (B_0 \vee B_1) \wedge \\ &\quad \mathit{def}\ B_0 \wedge \mathit{def}\ B_1 \quad . \end{aligned}$$

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

Array Assignment

- So far, all our arrays have been constants — we read from the arrays but never wrote to them!
- Consider $a : \textbf{array}\ [0..2)\ \textbf{of}\ \mathit{Int}$, 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,

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

- What went wrong?

- 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 $i = 2$?

- An array is a function. E.g. $a : \textbf{array } [0..N) \textbf{ of } \mathit{Bool}$ is a function $\mathit{Int} \to \mathit{Bool}$ 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?)

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

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

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

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

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

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

- 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.

- 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 $a[0] = 1 \wedge a[1] = 1$.

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

- 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.

Typical Array Manipulation in a Loop

Consider:

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

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

- The calculation can certainly be generalised.
- Given a function $H : Int \rightarrow A$, 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:

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

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

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

- 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.

Consider:

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

The Program

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

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

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

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

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

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

- 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.
- 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 $h[E]$.

- 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

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

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

- We have

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

- Thus we choose $\mathit{swap}\ h\ w\ (b - 1); b := b - 1$ 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 $r = w \vee h[r] = W$.

Red: Case $r = w$

- We have

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

- 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.

$\mathbf{con}\ K, N : Int\ \{0 \leqslant K < N\}$

$\mathbf{var}\ h : \mathbf{array}\ [0..N)\ \mathbf{of}\ A$

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

  rotation

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

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

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

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

- 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:

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

- 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:

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

- 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.

- 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:

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

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

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

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

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

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

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

  $SWAP\ U_1\ V$

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

  $U, B := U_0, U_1B$

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

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

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

  $SWAP\ U\ V_0$

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

  $A, V := AV_0, V_1$

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

  fi

- 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.

- Represented using indices:

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

- where $\mathit{SWAP}\ x\ \mathit{num}\ \mathit{off}$ abbreviates

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

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