Programming Languages:
Imperative Program Construction
9. Array Manipulation

Shin-Cheng Mu

Autumn Term, 2021

1 Some Notes on Definedness

Assignment Revisited

- Recall the weakest precondition for assignments:

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

- 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:

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

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

  $wp\ (\textbf{if}\ B_0 \to S_0\ |\ B_1 \to S_1)\ P =$
  $\quad B_0 \Rightarrow wp\ S_0\ P \wedge B_1 \Rightarrow wp\ S_1\ P \wedge (B_0 \vee B_1) \wedge$
  $\quad def\ B_0 \wedge def\ B_1\ .$

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 $A : \textbf{array}\ [M..N)\ \textbf{of}\ Int$ and let I be an expression. We define $def\ (A[I]) = def\ I \wedge M \leqslant I < N$.

- E.g. given $A : \textbf{array}\ [0..N)\ \textbf{of}\ Int$,

  - $def\ (A[x\ /\ z] + A[y]) = z \neq 0 \wedge 0 \leqslant x\ /\ z < N \wedge 0 \leqslant y < N$.
  - $wp\ (s := s \uparrow A[n])\ P = P[s \backslash s \uparrow A[n]] \wedge 0 \leqslant n < N$.

- We never made it explicit, because conditions such as $0 \leqslant n < N$ were usually already in the invariant/guard and thus discharged immediately.

2 Array Assignment

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

- Consider $a : \textbf{array}\ [0..2)\ \textbf{of}\ Int$, where $a[0] = 1$ and $a[1] = 1$.

- It should be true that

  $\{a[0] = 1 \wedge a[1] = 1\}$
  $a[a[1]] := 0$
  $\{a[a[1]] = 1\}\ .$

- However, if we use the previous wp,

$$wp\ (a[a[1]] := 0)\ (a[a[1]] = 1)$$
$$\equiv (a[a[1]] = 1)[a[a[1]] \backslash 0]$$
$$\equiv 0 = 1$$
$$\equiv \mathit{False}\ .$$

- 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. $a : \mathbf{array}\ [0..N)\ \mathbf{of}\ \mathit{Bool}$ 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?)

Function Alteration

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

$$(f : x \triangleright e)\ y = e \quad\ , \text{if } x = y;$$
$$= f\ y \quad , \text{otherwise.}$$

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

$$(f : 1 \triangleright -1)\ 1 = -1\ ,$$
$$(f : 1 \triangleright -1)\ x = x^2 \quad, \text{if } x \neq -1.$$

wp for Array Assignment

- Key: assignment to array should be understood as altering the entire function.

- Given $a : \mathbf{array}\ [M..N)\ \mathbf{of}\ A$ (for any type A), the updated rule:

$$wp\ (a[I] := E)\ P = P[a \backslash (a : I \triangleright E)]\ \wedge$$
$$\mathit{def}\ (a[I]) \wedge \mathit{def}\ E\ .$$

- In our examples, $\mathit{def}\ (a[I])$ and $\mathit{def}\ E$ 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

$$\{a[0] = 1 \wedge a[1] = 1\}$$
$$a[a[1]] := 0$$
$$\{a[a[1]] = 1\}\ .$$

- We aim to prove

$$a[0] = 1 \wedge a[1] = 1 \Rightarrow$$
$$wp\ (a[a[1]] := 0)\ (a[a[1]] = 1)\ .$$

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

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

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:

$$x, y := 0, 0;$$
$$a[x], a[y] := 0, 1$$

- 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:

> **con** $N : Int \; \{0 \leqslant N\}$
> **var** $h : $ **array** $[0..N)$ **of** Int
> *allzeros*
> $\{\langle \forall i : 0 \leqslant i < N : h[i] = 0 \rangle\}$

The Usual Drill

> **con** $N : Int \; \{0 \leqslant N\}$
> **var** $h : $ **array** $[0..N)$ **of** Int
> **var** $n : Int$
>
> $n := 0$
> $\{\langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle \wedge 0 \leqslant n \leqslant N,$
> $\quad bnd : N - n\}$
> **do** $n \neq N \to \; ?$
> $\qquad\qquad n := n + 1$
> **od**
> $\{\langle \forall i : 0 \leqslant i < N : h[i] = 0 \rangle\}$

Constructing the Loop Body

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

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

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

$$
\begin{aligned}
& \langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle \wedge 0 \leqslant n < N \Rightarrow \\
& \langle \forall i : 0 \leqslant i < n : (h : I \to E)[i] = 0 \rangle \wedge \\
& (h : I \to E)[n] = 0 \;.
\end{aligned}
$$

- An obvious choice: $(h : n \to 0)$,

- which almost immediately leads to

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

The Program

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

Obvious, but useful.

3.2 Simple Array Assignment

- The calculation can certainly be generalised.

- Given a function $H : Int \to A$, and suppose we want to establish

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

  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 $P \; n \wedge n \backslash = N$.

- One can prove the following:

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

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

> $\{P \; 0\}$
> $n := 0$
> $\{P \; n, bnd : N - n\}$
> **do** $n \neq N \to$
> $\qquad \{\text{establish } E = H \; n\}$
> $\quad h[n] := E$
> $\quad n := n + 1$
> **od**
> $\{\langle \forall i : 0 \leqslant i < N : h[i] = H \; i \rangle\}$

- 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:

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

The Up Loop Again

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

- A program skeleton:

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

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

$$\langle \forall i : 0 \leqslant i \leqslant 6 : h[i] = \langle \#k : 0 \leqslant k < 0 : X[k] = i \rangle \rangle$$
$$\equiv \langle \forall i : 0 \leqslant i \leqslant 6 : h[i] = 0 \rangle \ ,$$

- which can be performed by *allzeros*.

Constructing the Loop Body

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

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

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

$$\# \ False = 0$$
$$\# \ True \ = 1 \ .$$

- Again we conjecture that $h[I] := E$ will do the trick.

- We want to find I ane E such that $P\ n \wedge 0 \leqslant n < N \Rightarrow (P\ (n+1))[h \backslash (h : I \triangleright E)]$ can be proved.

- Assume $P\ n \wedge 0 \leqslant n < N$, consider $(P\ (n+1))[h \backslash (h : I \triangleright E)]$

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

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

The Program

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

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

4 Swaps

- Given array $h\ [0..N)$ and integer expressions E and F, we abbreviate the code fragment

$$[\![\ \textbf{var} \ r; r := h[E]; h[E] := h[F]; h[F] := r \ ]\!]$$

to *swap h E F*.

- |[...]| denotes a program block with local constants and variables. We have not used this feature so far.

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

Function Alteration

- We also extend the notion of function alteration to two entries.

$$(f : x, y \rightarrow e, f)\ z = e \quad \text{, if } z = x,$$
$$= f \quad \text{, if } z = y,$$
$$= f\ z \quad \text{, otherwise.}$$

- We have

$$wp\ (swap\ h\ E\ F)\ P = def\ (h[E]) \land def\ (h[F]) \land$$
$$P[h \backslash (h : E, F \rightarrow h[F], h[E])]\ .$$

Complications

- Note that it is *not* always the case that

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

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

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

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

A Simpler Case

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

$$(\{\langle \forall i : i \neq E \land i \neq F : h[i] = H\ i \rangle\} \land$$
$$h[E] = X \land h[E] = Y)$$
$$swap\ h\ E\ F$$
$$(\{\langle \forall i : i \neq E \land i \neq F : h[i] = H\ i \rangle\} \land$$
$$h[E] = Y \land h[E] = X)\ .$$

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

5 The Dutch National Flag

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

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

- The program shall manipulate h only by swapping.

- Denote the postcondition by Q.

Invariant

- Introduce a variable b.

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

$$P_0 \equiv P_r \land P_w \land P_b$$
$$P_1 \equiv 0 \leqslant r \leqslant w \leqslant b \leqslant N$$
$$P_r \equiv \langle \forall i : 0 \leqslant i < r :\ h[i] = R \rangle$$
$$P_w \equiv \langle \forall i : r \leqslant i < w :\ h[i] = W \rangle$$
$$P_b \equiv \langle \forall i : b \leqslant i < N : h[i] = B \rangle$$

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

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

The Plan

$$r, w, b := 0, 0, N$$
$$\{P_0 \land P_1, bnd : b - w\}$$
$$\textbf{do}\ b \neq w \rightarrow \textbf{if}\ h[w] = R\ \rightarrow S_r$$
$$\mid\ h[w] = W \rightarrow S_w$$
$$\mid\ h[w] = B\ \rightarrow S_b$$
$$\textbf{fi}$$
$$\textbf{od}$$
$$\{Q\}$$

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

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

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

Blue

- We have

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

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

Red

- Precondition: $P_r \land P_w \land P_b \land w < b \land h[w] = R$.

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

Red: Case $r = w$

- We have

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

Red: Case $h[r] = W$

- We have

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

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

The Program

$$\textbf{con}\ N : Int\ \{0 \leqslant N\}$$
$$\textbf{var}\ h : \textbf{array}\ [0..N)\ \textbf{of}\ RWB$$
$$\textbf{var}\ r, w, b : Int$$

$$r, w, b := 0, 0, N$$
$$\{P_0 \land P_1, bnd : b - w\}$$
$$\textbf{do}\ b \neq w \to \textbf{if}\ h[w] = R\ \ \to swap\ h\ w\ r$$
$$\qquad\qquad\qquad\qquad\qquad r, w := r+1, w+1$$
$$\qquad\quad |\ \ h[w] = W \to w := w+1$$
$$\qquad\quad |\ \ h[w] = B\ \ \to swap\ h\ w\ (b-1)$$
$$\qquad\qquad\qquad\qquad\qquad b := b-1$$
$$\qquad\quad \textbf{fi}$$
$$\textbf{od}$$
$$\{Q\}$$