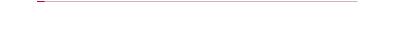# Programming Languages: Imperative Program Construction
9. Array Manipulation

Shin-Cheng Mu

Autumn Term, 2021

National Taiwan University and Academia Sinica

Some Notes on Definedness

- 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!

- 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* $(x + y \, / \, 2) = (2 \neq 0) = True$.

- A more complete rule:

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

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

$$wp\ (\text{if } B_0 \rightarrow S_0 \mid B_1 \rightarrow S_1 \text{ fi})\ P =$$
$$B_0 \Rightarrow wp\ S_0\ P \land B_1 \Rightarrow wp\ S_1\ P \land (B_0 \lor B_1) \land$$
$$def\ B_0 \land def\ B_1\ .$$

- 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 $A :$ **array** $[M..N)$ **of** Int and let I be an expression. We define $def\,(A[I]) = def\,I \land M \leqslant I < N$.
- E.g. given $A :$ **array** $[0..N)$ **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.

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

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

- However, if we use the previous wp,

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

- 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* : **array** [0..*N*) **of** *Bool* is a function *Int* → *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 $(f : x \to e)$ denote the function that *maps x to e*, and otherwise the same as *f*.

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

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

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

- 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:

$$wp\ (a[I] := E)\ P = P[a \setminus (a : I \rightarrow E)] \wedge$$
$$def\ (a[I]) \wedge def\ E\ .$$

- 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

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

- We aim to prove

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

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

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

- 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.

Typical Array Manipulation in a Loop

Consider:

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

con $N : Int \{0 \leqslant N\}$
var $h :$ **array** $[0..N)$ **of** Int
var $n : Int$

$n := 0$
$\{\langle \forall i : 0 \leqslant i < n : h[i] = 0 \rangle \land 0 \leqslant n \leqslant N,$
   $bnd : N - n\}$
do $n \neq N \rightarrow$?
                $n := n + 1$
od
$\{\langle \forall i : 0 \leqslant i < N : h[i] = 0 \rangle\}$

- 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 $P\ n \wedge n \setminus = N$.

- One can prove the following:

$$\{P \; n \land n \setminus = N \land E = H \; n\}$$
$$h[n] := E$$
$$\{P \; (n + 1)\} \quad ,$$

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

$$\{P\,0\}$$
$$n := 0$$
$$\{P\,n, bnd : N - n\}$$
do $n \neq N \rightarrow$
$$\quad \{\text{ establish } E = H\,n \}$$
$$h[n] := E$$
$$n := n + 1$$
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.

Consider:

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

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 : $ **array** $[0..N)$ **of** Int

$\{\langle \forall i : 0 \leqslant i < N : 1 \leqslant X[i] \leqslant 6 \rangle\}$

var $h : $ **array** $[1..6]$ **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

$\{\langle \forall i : 0 \leqslant i \leqslant 6 : h[i] =$

- 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.

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*
  > $\{0 \leqslant r \leqslant w \leqslant N \wedge$
  > $\langle \forall i : 0 \leqslant i < r : h[i] = R \rangle \wedge$
  > $\langle \forall i : r \leqslant i < w : h[i] = W \rangle \wedge$
  > $\langle \forall i : w \leqslant i < N : h[i] = B \rangle \wedge \}$

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

- 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

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

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

- 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

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