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:

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

- 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 $\mathit{def}\ E$.
- 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.

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

- 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 : **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?

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

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

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

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

- 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.

The Example

- Recall our example

$$
\begin{aligned}
&\{a[0] = 1 \wedge a[1] = 1\} \\
&a[a[1]] := 0 \\
&\{a[a[1]] = 1\} \quad .
\end{aligned}
$$

- We aim to prove

$$
\begin{aligned}
&a[0] = 1 \wedge a[1] = 1 \Rightarrow \\
&\quad wp\ (a[a[1]] := 0)\ (a[a[1]] = 1) \quad .
\end{aligned}
$$

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

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

- 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:

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

- 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:

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

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

- The calculation can certainly be generalised.
- Given a function $H : Int \to A$, 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 $P\ n \wedge n \setminus = N$.

- One can prove the following:

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

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

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

- 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.

Consider:

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

The Program

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

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

$$n := 1$$
$$\textbf{do}\ n \neq 7 \rightarrow h[n] := 0; n := n + 1\ \textbf{od}$$
$$\{P\ 0\}$$

$$n := 0$$
$$\{P\ n \wedge 0 \leqslant n \leqslant N, bnd : N - n\}$$
$$\textbf{do}\ n \neq N \rightarrow h[X[n]] := h[X[n]] + 1$$
$$n := n + 1$$
$$\textbf{od}$$
$$\{\langle \forall i : 0 \leqslant i \leqslant 6 : h[i] =$$
$$\langle \# k : 0 \leqslant k < N : X[k] = i \rangle \rangle\}$$

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

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

- 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).

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

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

- The case for white is the easiest, since

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

- 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.

- 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

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