Warm-Up Problem

What are some sample loop invariants of the following piece of code?

$(\!|\ ((y = y_0) \land (y \geq 0))\ |\!)$

```
z = 0 ;
while ( y > 0 ) {
    z = z + x ;
    y = y - 1 ;
}
```

$(\!|\ ???\ |\!)$

Program Verification
Arrays

Carmen Bruni

Lecture 20

Based on slides by Jonathan Buss, Lila Kari, Anna Lubiw and Steve Wolfman with thanks to B. Bonakdarpour, A. Gao, D. Maftuleac, C. Roberts, R. Trefler, and P. Van Beek

Last Time

- Partial correctness for while loops
- Determine whether a given formula is an invariant for a while loop.
- Find an invariant for a given while loop.
- Prove that a Hoare triple is satisfied under partial correctness for a program containing while loops.

Learning Goals

- Introducing the array assignment rule
- Annotate code using this rule
- Prove that a Hoare triple is satisfied under partial correctness for a program containing array assignment statments.

Assignment of Values of an Array

Let `A` be an array of `n` integers: `A[1]`, `A[2]`, ..., `A[n]`.

Assignment may work as before:

$(\!|\ P[A[x]/v]\ |\!)$
`v = A[x] ;`
$(\!|\ P\ |\!)$ assignment

But a complication can occur:

$(\!|\ A[y] = 0\ |\!)$
`A[x] = 1;`
$(\!|\ A[y] = 0\ |\!)$???

The conclusion is not valid if $x = y$.

A correct rule must account for possible changes to `A[y]`, `A[z+3]`, etc., when `A[x]` changes.

Assignment to a Whole Array

Our solution: Treat an assignment to an array value

$$\texttt{A[e}_1\texttt{] = e}_2$$

as an assignment of the whole array:

$$\texttt{A = A}\{\texttt{e}_1 \leftarrow \texttt{e}_2\}\ \texttt{;}$$

where the term "$\texttt{A}\{\texttt{e}_1 \leftarrow \texttt{e}_2\}$" denotes an array identical to `A` except the $\texttt{e}_1^{th}$ element is changed to have the value $\texttt{e}_2$.

Array Assignment: Definition and Examples

Definition: $A\{i \leftarrow e\}$ denotes the array with entries given by

$$A\{i \leftarrow e\}[j] = \begin{cases} e, & \text{if } j = i \\ A[j], & \text{if } j \neq i \end{cases} .$$

Examples:

1. $A\{1 \leftarrow 3\}[1] = 3$
2. $A\{1 \leftarrow 3\}\{1 \leftarrow 4\}[1] = 4$

The Array-Assignment Rule

Array assignment:

$$\frac{}{\llparenthesis\, Q[A\{e_1 \leftarrow e_2\}/A] \,\rrparenthesis \;\; \texttt{A[e}_1\texttt{] = e}_2 \;\; \llparenthesis\, Q \,\rrparenthesis} \text{(Array assignment)}$$

where

$$A\{i \leftarrow e\}[j] = \begin{cases} e, & \text{if } j = i \\ A[j], & \text{if } j \neq i \end{cases} .$$

Example

Prove the following is satisfied under partial correctness.

$$\llparenthesis \, ((A[x] = x_0) \wedge (A[y] = y_0)) \, \rrparenthesis$$

```
t = A[x] ;
A[x] = A[y] ;
A[y] = t ;
```

$$\llparenthesis \, ((A[x] = y_0) \wedge (A[y] = x_0)) \, \rrparenthesis$$

We do assignments bottom-up, as always....

Example: push up assertions for assignments

$(\!|\ ((A[x] = x_0) \wedge (A[y] = y_0))\ |\!)$

```
t = A[x] ;
```

```
A[x] = A[y] ;
```

$(\!|\ ((A\{y \leftarrow t\}[x] = y_0) \wedge (A\{y \leftarrow t\}[y] = x_0))\ |\!)$

```
A[y] = t ;
```

$(\!|\ ((A[x] = y_0) \wedge (A[y] = x_0))\ |\!)$ array assignment

Example: push up assertions for assignments

$$\llparenthesis\, ((A[x] = x_0) \wedge (A[y] = y_0))\, \rrparenthesis$$

```
t = A[x] ;
```

$$\llparenthesis\, ((A\{x \leftarrow A[y]\}\{y \leftarrow t\}[x] = y_0) \wedge (A\{x \leftarrow A[y]\}\{y \leftarrow t\}[y] = x_0))\, \rrparenthesis$$

```
A[x] = A[y] ;
```

$$\llparenthesis\, ((A\{y \leftarrow t\}[x] = y_0) \wedge (A\{y \leftarrow t\}[y] = x_0))\, \rrparenthesis$$ array assignment

```
A[y] = t ;
```

$$\llparenthesis\, ((A[x] = y_0) \wedge (A[y] = x_0))\, \rrparenthesis$$ array assignment

Example: push up assertions for assignments

$(\!| ((A[x] = x_0) \wedge (A[y] = y_0)) |\!)$

$(\!| ((A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[x] = y_0) \wedge (A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[y] = x_0)) |\!)$

```
t = A[x] ;
```

$(\!| ((A\{x \leftarrow A[y]\}\{y \leftarrow t\}[x] = y_0) \wedge (A\{x \leftarrow A[y]\}\{y \leftarrow t\}[y] = x_0)) |\!)$ — assignment

```
A[x] = A[y] ;
```

$(\!| ((A\{y \leftarrow t\}[x] = y_0) \wedge (A\{y \leftarrow t\}[y] = x_0)) |\!)$ — array assignment

```
A[y] = t ;
```

$(\!| ((A[x] = y_0) \wedge (A[y] = x_0)) |\!)$ — array assignment

Example: push up assertions for assignments

$$(\!|\ ((A[x] = x_0) \wedge (A[y] = y_0))\ |\!)$$

$$(\!|\ ((A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[x] = y_0) \wedge (A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[y] = x_0))\ |\!)$$

implied (a)

```
t = A[x] ;
```

$$(\!|\ ((A\{x \leftarrow A[y]\}\{y \leftarrow t\}[x] = y_0) \wedge (A\{x \leftarrow A[y]\}\{y \leftarrow t\}[y] = x_0))\ |\!)$$

assignment

```
A[x] = A[y] ;
```

$$(\!|\ ((A\{y \leftarrow t\}[x] = y_0) \wedge (A\{y \leftarrow t\}[y] = x_0))\ |\!)$$

array assignment

```
A[y] = t ;
```

$$(\!|\ ((A[x] = y_0) \wedge (A[y] = x_0))\ |\!)$$

array assignment

Example: Proof of implied

As "implied (a)", we need to prove the following.

Lemma:

$$(A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[x] = A[y])$$

and

$$(A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[y] = A[x]) \ .$$

Proof.

In the second equation, the index element is the assigned element.

For the first equation, we consider two cases.

- If $y \neq x$, the "$\{y \leftarrow \dots\}$" is irrelevant, and the claim holds.
- If $y = x$, the result on the left is $A[x]$, which is also $A[y]$.

Example: Alternative proof

For an alternative proof, use the definition of $M\{i \leftarrow e\}[j]$, with $A\{x \leftarrow A[y]\}$ as M, $i = y$ and $e = A[x]$:

$$A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[j] = \begin{cases} A[x], & \text{if } y = j \\ A\{x \leftarrow A[y]\}[j], & \text{if } y \neq j \end{cases} .$$

At index $j = y$, this is just $A[x]$, as required.

In the case $j = x$, we get the required value $A[y]$. (Why?)

And, finally, if $j \neq x$ and $j \neq y$, then

$$A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[j] = A[j] \ ,$$

as we should have required.

Example: reversing an array

Example: Given an array R with n elements $R[1], \ldots, R[n]$, reverse the elements.

Algorithm: exchange $R[j]$ with $R[n+1-j]$, for each $1 \leq j \leq \lfloor n/2 \rfloor$.

A possible program is

```
j = 1 ;
while ( 2*j <= n ) {
    t = R[j] ;
    R[j] = R[n+1-j] ;
    R[n+1-j] = t ;
    j = j + 1 ;
}
```

Needed: a postcondition, and a loop invariant.

Reversal code: conditions and an invariant

Precondition: $\Big(\forall x \left((1 \le x \le n) \to (R[x] = r_x)\right)\Big)$.

Postcondition: $\Big(\forall x \left((1 \le x \le n) \to (R[x] = r_{n+1-x})\right)\Big)$.

Invariant? When has an exchange occurred at position x?

- If $x < j$ or $x > n+1-j$, then $R[x]$ and $R[n+1-x]$ have already been exchanged.
- If $j \le x \le n+1-j$, then no exchange has happened yet.

Thus let *Inv'*(j) be the formula

$$\Bigg(\forall x \Bigg(\Big((1 \le x < j) \to (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x)\Big) \wedge \Big((j \le x \le (n+1)/2) \to (R[x] = r_x \wedge R[n+1-x] = r_{n+1-x})\Big)\Bigg)\Bigg) .$$

and *Inv*$(j) =$ *Inv'*$(j) \wedge (1 \le j \le n/2+1)$.

Question

Why can we not just use

$$\left(\forall x \left((1 \leq x < j) \rightarrow (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x)\right)\right) \; .$$

as our invariant $\mathrm{Inv}'(j)$? More on this later...

Reversal: Annotations around the loop

The annotations surrounding the `while`-loop:

$(\!|\ \big((n > 0) \land \big(\forall x \,\big((1 \leq x \leq n) \to (R[x] = r_x)\big)\big)\big)\ |\!)$

$(\!|\ Inv(1)\ |\!)$ — implied (a)

```
j = 1 ;
```

$(\!|\ Inv(j)\ |\!)$ — assignment

```
while ( 2*j <= n ) {
```

$(\!|\ (Inv(j) \land (2j \leq n))\ |\!)$ — partial-`while`

⋮

$(\!|\ Inv(j)\ |\!)$ — *(TBA)*

```
}
```

$(\!|\ (Inv(j) \land (2j > n))\ |\!)$ — partial-`while`

$(\!|\ \big(\forall x \,\big((1 \leq x \leq n) \to (R[x] = r_{n+1-x})\big)\big)\ |\!)$ — implied (b)

Reversal code: annotations inside the loop

We must now handle the code inside the loop.

$(\!|\ (Inv(j) \land 2j \leq n)\ |\!)$ partial-while

$(\!|\ Inv(j+1)[R'/R]$, *where* R' *is*
$R\{j \leftarrow R[n+1-j]\}\{(n+1-j) \leftarrow R[j]\}\ |\!)$ implied (c)

```
t = R[j]; R[j] = R[n+1-j]; R[n+1-j] = t;
```

$(\!|\ Inv(j+1)\ |\!)$ Lemma

```
j = j + 1 ;
```

$(\!|\ Inv(j)\ |\!)$ assignment

Full Annotation

$\llparenthesis\, ((n > 0) \wedge (\forall x\, (((1 \leq x) \wedge (x \leq n)) \rightarrow (R[x] = r_x))))\, \rrparenthesis$

$\llparenthesis\, (\mathrm{Inv}'(1) \wedge (1 \leq (\frac{n}{2} + 1)))\, \rrparenthesis$ — Implied(a)

`j = 1 ;`

$\llparenthesis\, (\mathrm{Inv}'(j) \wedge (j \leq (\frac{n}{2} + 1)))\, \rrparenthesis$ — Assignment

`while (2 * j <= n) {`

    $\llparenthesis\, ((\mathrm{Inv}'(j) \wedge (j \leq (\frac{n}{2} + 1))) \wedge ((2 \cdot j) <= n))\, \rrparenthesis$ — Partial-While

    $\llparenthesis\, (\mathrm{Inv}'((j + 1))[R'/R] \wedge ((j + 1) \leq (\frac{n}{2} + 1)))\, \rrparenthesis$ — Implied(c)

    `t = R[j]; R[j] = R[n+1-j]; R[n+1-j] = t;`

    $\llparenthesis\, (\mathrm{Inv}'((j + 1)) \wedge ((j + 1) \leq (\frac{n}{2} + 1)))\, \rrparenthesis$ — Lemma

    `j = j + 1 ;`

    $\llparenthesis\, (\mathrm{Inv}'(j) \wedge (j \leq (\frac{n}{2} + 1)))\, \rrparenthesis$ — Assignment

`}`

$\llparenthesis\, ((\mathrm{Inv}'(j) \wedge (j \leq (\frac{n}{2} + 1))) \wedge ((2 \cdot j) > n))\, \rrparenthesis$ — Partial-While

$\llparenthesis\, (\forall x\, (((1 \leq x) \wedge (x \leq n)) \rightarrow (R[x] = r_{n+1-x})))\, \rrparenthesis$ — Implied(b)

Question

Why can we not just use

$$\left(\forall x\left((1 \le x < j) \to (R[x] = r_{\boldsymbol{n+1-x}} \wedge R[n+1-x] = r_{\boldsymbol{x}})\right)\right) \ .$$

as our invariant $\mathrm{Inv}'(j)$?

Question

Why can we not just use

$$\left(\forall x \left((1 \leq x < j) \rightarrow (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x)\right)\right) .$$

as our invariant $\mathrm{Inv}'(j)$? The implication Implied (c) Is actually unproveable! If we don't know the starting values of the array, we won't be able to prove the situation when $j = 1$. (Try it!)

Remains To Show

- It remains to prove all of the implied conditions on the previous slide are true (including the lemma).
- We also should complete the prof of total correctness by showing that the while loop terminates.
- We prove the latter first.

Proof that the While Loop Terminates

Consider the loop variant

$$V = \lfloor \frac{n}{2} \rfloor + 1 - j$$

This variant is non-negative and on each iteration of the loop, n remains unchanged while j increments by one. Hence the variant above decrements by 1 each time. Thus, $V \geq 0$ at the beginning since $j = 1$ [note $n > 0$ could be added as well; without this, there would be no elements in the array] and decreases by one during each loop.

Our loop guard is $2j \leq n$ or reworded $j \leq \lfloor \frac{n}{2} \rfloor$. When $V = 0$, we have that $j = \lfloor \frac{n}{2} \rfloor + 1 > \lfloor \frac{n}{2} \rfloor$ and thus the while loop terminates. Hence, our code must terminate.

Proof of Lemma

Why is the lemma true?

Proof of Lemma

Why is the lemma true?

The justification for the Lemma is that the three assignment lines simply **swap** the entries $R[j]$ and $R[n+1-j]$ (by the earlier "baby" example of verifying a single swap), and the usual approach to constructing a correct precondition from a given post-condition, with an assignment between.

Proof of Implied (a)

Implied (a) is

$$\left((\forall x\, ((1 \le x \le n) \rightarrow (R[x] = r_x))) \rightarrow \left(\mathrm{Inv}'(1) \wedge \left(1 \le \frac{n}{2} + 1 \right) \right) \right).$$

Proof of Implied (a)

Implied (a) is

$$\left((\forall x\, ((1 \le x \le n) \to (R[x] = r_x))) \to \left(\mathrm{Inv}'(1) \wedge \left(1 \le \frac{n}{2} + 1\right)\right)\right).$$

$\mathrm{Inv}'(1)$ reads

$$\begin{aligned}&(\forall x\, ((1 \le x < 1) \to ((R[x] = r_{n+1-x}) \wedge (R[n+1-x] = r_x)))\\ &\wedge \left(\left(1 \le x \le \tfrac{n+1}{2}\right) \to ((R[x] = r_x) \wedge (R[n+1-x] = r_{n+1-x}))\right)).\end{aligned}$$

Proof of Implied (a)

Implied (a) is

$$\left((\forall x\,((1 \le x \le n) \to (R[x] = r_x))) \to \left(\mathrm{Inv}'(1) \land \left(1 \le \frac{n}{2} + 1\right)\right)\right).$$

$\mathrm{Inv}'(1)$ reads

$$\begin{array}{l}(\forall x\,((1 \le x < 1) \to ((R[x] = r_{n+1-x}) \land (R[n+1-x] = r_x))) \\ \land\,(\left(1 \le x \le \frac{n+1}{2}\right) \to ((R[x] = r_x) \land (R[n+1-x] = r_{n+1-x})))).\end{array}$$

Since no x can satisfy $(1 \le x < 1)$, there is nothing to check in the first implication. The second implication is simply the given precondition re-written, and so the required implication holds.

Proof of Implied (c)

Implied (c) is $\Big(((\mathrm{Inv}'(j) \wedge (j \leq (\frac{n}{2}+1))) \wedge ((2 \cdot j) <= n))$

$$\rightarrow (\mathrm{Inv}'((j+1))[R'/R] \wedge ((j+1) \leq (\frac{n}{2}+1)))\Big)$$

Proof of Implied (c)

Implied (c) is $\Big(((\mathrm{Inv}'(j) \wedge (j \leq (\frac{n}{2}+1))) \wedge ((2 \cdot j) <= n))$

$$\rightarrow (\mathrm{Inv}'((j+1))[R'/R] \wedge ((j+1) \leq (\frac{n}{2}+1)))\Big)$$

By construction, R' and R are identical, except at indices j and $n+1-j$. So $\mathrm{Inv}'(j)$ will imply $\mathrm{Inv}'(j+1)[R'/R]$, provided everything is correct in R' at these indices. Everything is clear from the definitions, except possibly the entries $x = j$ and $x = n+1-j$.

- For $(x = j)$ hypothesis is $((R[j] = r_j) \wedge (R[n+1-j] = r_{n+1-j}))$
- For $(x = j)$ conclusion is $((R'[j] = r_{n+1-j}) \wedge (R'[n+1-j] = r_j))$
- By the definition for R', we have

$$\begin{aligned} R'[j] &= R[n+1-j] = r_{n+1-j}, \text{ and} \\ R'[n+1-j] &= R[j] = r_j, \end{aligned}$$

  which completes the proof.

Proof of Implied (b)

Recall that Implied (b) is

$$\Big(\Big(\Big(\mathrm{Inv}'(j) \wedge \Big(j \leq (\tfrac{n}{2}+1)\Big)\Big) \wedge ((2 \cdot j) > n)\Big)$$
$$\rightarrow (\forall x\,((1 \leq x \leq n) \rightarrow (R[x] = r_{n+1-x})))\Big).$$

Proof of Implied (b)

Recall that Implied (b) is

$$\Big(\Big(\Big(\mathrm{Inv}'(j) \wedge \Big(j \leq (\tfrac{n}{2}+1)\Big)\Big) \wedge ((2 \cdot j) > n)\Big) \\ \rightarrow (\forall x\,((1 \leq x \leq n) \rightarrow (R[x] = r_{n+1-x})))\Big).$$

Recall that $\mathrm{Inv}'(j)$ reads

$$(\forall x\,(1 \leq x < j \rightarrow (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x)) \\ \wedge (j \leq x \leq \tfrac{n+1}{2} \rightarrow (R[x] = r_x \wedge R[n+1-x] = r_{n+1-x}))).$$

We must analyze $(\Big(j \leq \frac{n}{2}+1\Big) \wedge (2 \cdot j > n))$ for the cases where n is even and odd.

Case n is even.

If n is even, then $\frac{n}{2}$ is an integer, so that $((j \leq \frac{n}{2} + 1) \wedge (2 \cdot j > n))$ gives $j = \frac{n}{2} + 1$. Now $\mathrm{Inv}'\left(\frac{n}{2} + 1\right)$ reads

$$(\forall x\,(1 \leq x < \tfrac{n}{2} + 1 \rightarrow (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x))$$
$$\wedge\,(\tfrac{n}{2} + 1 \leq x \leq \tfrac{n+1}{2} \rightarrow (R[x] = r_x \wedge R[n+1-x] = r_{n+1-x}))).$$

Clearly, no x can satisfy the hypothesis of the second implication, so there is nothing to check there. We may rewrite the first implication as

$$(1 \leq x \leq \frac{n}{2} \rightarrow (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x)).$$

This asserts that all the required swaps have been performed, and so the required program post-condition holds, and we are done in this case.

Case n is odd.

If n is odd, then $\frac{n+1}{2}$ is an integer, so that $(\left(j \leq \frac{n}{2}+1\right) \wedge (2 \cdot j > n))$, equivalently $(\left(j \leq \frac{n+1}{2}+\frac{1}{2}\right) \wedge (2 \cdot j > n))$ gives $j = \frac{n+1}{2}$. Now $\mathrm{Inv}'\left(\frac{n+1}{2}\right)$ reads

$$\begin{array}{l}(\forall x\,(1 \leq x < \frac{n+1}{2} \rightarrow (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x)) \\ \wedge\,(\frac{n+1}{2} \leq x \leq \frac{n+1}{2} \rightarrow (R[x] = r_x \wedge R[n+1-x] = r_{n+1-x}))).\end{array}$$

Only $x = \frac{n+1}{2}$ can satisfy the hypothesis of the second implication. Both halves of the $\wedge$-formula in the conclusion of the implication then assert that the middle element was not changed, because $n+1-\frac{n+1}{2} = \frac{n+1}{2}$. We may rewrite the first implication as

$$(1 \leq x \leq \frac{n-1}{2} \rightarrow (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x)).$$

This asserts that all the required swaps have been performed, and so the required program post-condition holds, and so we are done in this case.