Array Assignments

Stefan Muller
based on material by Jim Sasaki

CS 536: Science of Programming, Fall 2023
Lecture 18

1 Introduction

Up until now, we haven't talked about Hoare triples, weakest preconditions, etc., for programs with array assignments. This seems odd: after all, on first glance, array assignments are much more straightforward than something like loops, which we also deferred talking about. Why can't we just use the same rules as for normal assignments? Consider the program and postcondition

$$S \triangleq x := y; y := y + \overline{1} \qquad Q \triangleq x < y$$

The weakest liberal precondition $wlp(S, Q)$ would be

$$[y/x][y + 1/y]Q = y < y + 1 \Leftrightarrow T$$

Now consiser a similar program, but on array elements $a[k]$ and $a[j]$ instead of variables x and y.

$$S \triangleq a[k] := a[j]; a[j] := a[j] + \overline{1} \qquad Q \triangleq a[k] < a[j]$$

What should $wlp(a[j] := a[j] + \overline{1}, a[k] < a[j])$ be? Well, let's do the normal substitution. We get $[a[j] + 1/a[j]]a[k] < a[j]$. But what is this substitution? It seems obvious but what if $k = j$? In this case, $a[k]$ and $a[j]$ are the same element, and we should substitute $a[j] + 1$ for $a[k]$ too, to get $a[j] + 1 < a[j] + 1$ (which is, of course, false).

The problem is that we don't know when we're trying to prove things about the program what the values of array indices are, and therefore changing one array element might also change another array element (because they're actually the same at runtime). This is an instance of a broader problem known as *aliasing*, which we'll talk about more later.

Today, we'll extend the definition of substitution to handle substituting for array elements, which will allow us to define Hoare logic rules and weakest preconditions for array assignments.

2 Substitution for Array Assignments

Most cases of substituting for array elements in expressions and predicates are straightforward because they just propagate the substitution (e.g., $[e/a[e_1]](x := e_2) = x := [e/a[e_1]]e_2$, as before). The tricky case is when we're substituting for an array element in an array element:

$$[e/a[e_1]]b[e_2]$$

Here, a and b may refer to the same array or different arrays.

There's a general rule, but let's consider some (easier) special cases first.

Case 1: Different arrays

If a and b are different arrays (we'll know this syntactically because the a in $a[e]$ isn't an expression, it has to be the actual name of the array), we don't have to worry about $a[e_1]$ and $b[e_2]$ being the same element, so we can just continue substituting:

$$[e/a[e_1]]b[e_2] = b[[e/a[e_1]]e_2]$$

Of course, this may still require us to handle trickier cases later if a appears in e_2.

Case 2: Same array, constant or variable index

Next, we consider the case

$$[e/a[e_1]]a[k]$$

where k is a constant or variable.

If $k = e_1$, then we want to do the substitution. Otherwise, we don't. How do we know whether $k = e_1$? We don't until we run the program. But we can produce an expression that does the check at runtime, using a conditional expression.

$$[e/a[e_1]]a[k] = \mathsf{if}\ k = e_1\ \mathsf{then}\ e\ \mathsf{else}\ a[k]$$

Examples:

1. $[5/a[0]]a[k] = \mathsf{if}\ k = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[k]$
2. $[e/a[j]]a[k] = \mathsf{if}\ k = j\ \mathsf{then}\ e\ \mathsf{else}\ a[k]$
3. $[a[j] + 1/a[j]]a[k] = \mathsf{if}\ k = j\ \mathsf{then}\ a[j] + 1\ \mathsf{else}\ a[k]$
4. $[a[i]/a[j]]a[k] = \mathsf{if}\ k = j\ \mathsf{then}\ a[i]\ \mathsf{else}\ a[k]$

Case 2*: Same array, general index

If the index we're substituting into isn't a constant or variable, we need to substitute into *it* also, before checking for equality:

$$[e/a[e_1]]a[e_2] = \mathsf{if}\ e_2' = e_1\ \mathsf{then}\ e\ \mathsf{else}\ a[e_2'] \text{ where } e_2' = [e/a[e_1]]e_2$$

This subsumes Case 2, since if $e_2 = k$, then $e_2' = k$.

Example. Consider $[5/a[0]]a[a[k]]$. Following the definition, we get

$$[5/a[0]]a[a[k]] = \mathsf{if}\ e_2' = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[e_2']$$

where $e_2' = [5/a[0]]a[k] = \mathsf{if}\ k = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[k]$. So, we have

$$\mathsf{if}\ (\mathsf{if}\ k = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[k]) = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[\mathsf{if}\ k = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[k]]$$

This makes sense: if $k = 0$, then $a[k]$ is the same as $a[0]$, so $a[a[k]]$ is $a[5]$. But if $k \neq 0$ and $a[k] = 0$, then $a[a[k]]$ is $a[0]$. We'll see in the next section how to simplify expressions like this.

3 Hoare Triples and Weakest Precondition

With this new definition of substitution, we can give rules for proving Hoare Triples and computing weakest preconditions that look like the rules for normal assignments, but substitute for array elements.

$$\frac{}{\vdash \{[e/a[i]]Q\}\ a[i] := e\ \{Q\}}\ (\textsc{ArrAssignBwd})$$

$$wlp(a[e_1] := e_2, Q) = [e_2/a[e_1]]Q$$

4 Static Optimization

As you've seen, these substitution rules can result in pretty complicated expressions. Sometimes, we can simplify these "statically", that is, before running the program. For example, if our expression is

$$[5/a[1]]a[0] = \mathsf{if}\ 0 = 1\ \mathsf{then}\ 5\ \mathsf{else}\ a[0]$$

We know without running the program that $0 \neq 1$, so we can just simplify the above to $a[0]$.

We write $e_1 \Rrightarrow e_2$ to say that e_1 "simplifies to" e_2.

In general, if we know that $k = e$, then

$$\mathsf{if}\ k = e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Rrightarrow e_1$$

and if we know that $k \neq e$, then

$$\mathsf{if}\ k = e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Rrightarrow e_2$$

Example. We can sometimes do this with non-constants. For example:

$$[e/a[k]]a[k] = \mathsf{if}\ k = k\ \mathsf{then}\ e\ \mathsf{else}\ a[k] \Rrightarrow e$$

But, we can't assume for any k and j that $k \neq j$, because they could have the same value at runtime.

$$[e/a[k]]a[j] = \mathsf{if}\ k = j\ \mathsf{then}\ e\ \mathsf{else}\ a[j] \not\Rrightarrow a[j]$$

Even when we don't know statically whether two expressions are equal, we can still sometimes simplify expressions. Take an example from earlier:

$$\mathsf{if}\ (\mathsf{if}\ k = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[k]) = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[\mathsf{if}\ k = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[k]]$$

We don't know whether $k = 0$, but we do know that k doesn't change, so both times we condition on $k = 0$, the answer will be the same, and we can simplify this to

$$\mathsf{if}\ k = 0\ \mathsf{then}\ a[5]\ \mathsf{else}\ (\mathsf{if}\ a[k] = 0\ \mathsf{then}\ 5\ \mathsf{else}\ a[a[k]])$$

Rules for Simplifying Expressions

$$\begin{array}{rcll} \mathsf{if}\ T\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 & \Rrightarrow & e_1 & \text{Always} \\ \mathsf{if}\ e_0\ \mathsf{then}\ e\ \mathsf{else}\ e & \Rrightarrow & e & \text{Always} \\ \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 & \Rrightarrow & e_2 & \text{If } e_0 \Rightarrow e_1 = e_2 \end{array} \qquad \begin{array}{rcll} \mathsf{if}\ F\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 & \Rrightarrow & e_2 & \text{Always} \\ & & & \\ \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 & \Rrightarrow & e_1 & \text{If } \neg e_0 \Rightarrow e_1 = e_2 \end{array}$$

Let Θ be a unary operator, $\oplus$ be a binary operator or relation and f be any function.

$$\begin{array}{rcl} \Theta(\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) & \Rrightarrow & \mathsf{if}\ e\ \mathsf{then}\ \Theta(e_1)\ \mathsf{else}\ \Theta(e_2) \\ a[\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] & \Rrightarrow & \mathsf{if}\ e\ \mathsf{then}\ a[e_1]\ \mathsf{else}\ a[e_2] \end{array} \qquad \begin{array}{rcl} (\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) \oplus e_3 & \Rrightarrow & \mathsf{if}\ e\ \mathsf{then}\ e_1 \oplus e_3\ \mathsf{else}\ e_2 \oplus e_3 \\ f(\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) & \Rrightarrow & \mathsf{if}\ e\ \mathsf{then}\ f(e_1)\ \mathsf{else}\ f(e_2) \end{array}$$

If e, e_1, and e_2 are Boolean expressions, then

$$\begin{array}{rcl} (\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ F) & \Leftrightarrow & (e \wedge e_1) \\ (\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ T) & \Leftrightarrow & (e \to e_1) \Leftrightarrow (\neg e \vee e_1) \end{array} \qquad \begin{array}{rcl} (\mathsf{if}\ e\ \mathsf{then}\ F\ \mathsf{else}\ e_2) & \Leftrightarrow & (\neg e \wedge e_2) \\ (\mathsf{if}\ e\ \mathsf{then}\ T\ \mathsf{else}\ e_2) & \Leftrightarrow & (\neg e \to e_2) \Leftrightarrow (e \vee e_2) \end{array}$$

$$(\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) \Leftrightarrow ((e \to e_1) \wedge (\neg e \to e_2)) \Leftrightarrow ((e \wedge e_1) \vee (\neg e \wedge e_2))$$

Example.

$$\begin{array}{rl} & wlp(a[j] := a[j] + \overline{1}, a[k] < a[j]) \\ = & [a[j] + 1/a[j]](a[k] < a[j]) \\ = & [a[j] + 1/a[j]]a[k] < [a[j] + 1/a[j]]a[j] \\ = & (\mathsf{if}\ k = j\ \mathsf{then}\ a[j] + 1\ \mathsf{else}\ a[k]) < (\mathsf{if}\ j = j\ \mathsf{then}\ a[j] + 1\ \mathsf{else}\ a[k]) \\ \Rrightarrow & (\mathsf{if}\ k = j\ \mathsf{then}\ a[j] + 1\ \mathsf{else}\ a[k]) < a[j] + 1 \\ \Leftrightarrow & \mathsf{if}\ k = j\ \mathsf{then}\ (a[j] + 1 < a[j] + 1)\ \mathsf{else}\ (a[k] < a[j] + 1) \\ \Leftrightarrow & \mathsf{if}\ k = j\ \mathsf{then}\ F\ \mathsf{else}\ (a[k] < a[j] + 1) \\ \Leftrightarrow & k \neq j \wedge a[k] < a[j] + 1 \end{array}$$

5 Example: Swapping Array Elements

Remember we proved the following triple a while ago:

$$\{x = a \wedge y = b\}\ t := x; x := y; y := t\ \{x = b \wedge y = a\}$$

Now let's do the same with array assignments:

$$\{a[k] = c \wedge a[j] = d\}\ t := a[k]; a[k] := a[j]; a[j] := t\ \{a[k] = d \wedge a[j] = c\}$$

$$\begin{array}{ll}
 & \{a[k] = c \wedge a[j] = d\} \\
 & \Rightarrow \{[a[k]/t]((\mathsf{if}\ k = j\ \mathsf{then}\ t = d\ \mathsf{else}\ a[j] = d) \wedge t = c)\} \\
 & = \{(\mathsf{if}\ k = j\ \mathsf{then}\ a[k] = d\ \mathsf{else}\ a[j] = d) \wedge a[k] = c\} \\
 & = \{a[j] = d \wedge a[k] = c\} \\
t := a[k]; & \{[a[j]/a[k]]((\mathsf{if}\ k = j\ \mathsf{then}\ t = d\ \mathsf{else}\ a[k] = d) \wedge t = c) = (\mathsf{if}\ k = j\ \mathsf{then}\ t = d\ \mathsf{else}\ a[j] = d) \wedge t = c\} \\
a[k] := a[j]; & \{[t/a[j]](a[k] = d \wedge a[j] = c) = (\mathsf{if}\ k = j\ \mathsf{then}\ t = d\ \mathsf{else}\ a[k] = d) \wedge t = c\} \\
a[j] := t & \{a[k] = d \wedge a[j] = c\}
\end{array}$$