

# Array Element Assignments

CS 536: Science of Programming, Fall 2021

## A. Why?

- Array assignments aren't like assignments to plain variables because the actual item to change can't be determined until runtime. We can handle this by extending our notion of assignment and/or substitution.

## B. Outcomes

After this class, you should

- Know how to perform textual substitution to replace an array element.
- Know how to calculate the  $wp$  of an array element assignment.

## C. Array Element Assignments

- Up to now, we haven't covered Hoare triples,  $wp$ ,  $sp$ , etc., involving assignment to arrays.
- An array assignment  $a[e_0] := e_1$  (where  $e_0$  and  $e_1$  are expressions) is different from a plain variable assignment because the exact element being changed may not be known at program annotation time. E.g., compare these two triples:

- *Valid:*  $\{T\} x := y; y := y+1 \{x < y\}$
- *Invalid:*  $\{T\} a[k] := a[j]; a[j] := a[j]+1 \{a[k] < a[j]\}$

- The problem is what happens if  $k = j$  at runtime: What is

$wp(a[j] := a[j]+1, a[k] < a[j])$  ?

- The answer should be something like

" $k \neq j$  ?  $a[k] < a[j]+1$  :  $a[j]+1 < a[j]+1$  (which is false)"

- There are two alternatives for handling array assignments
- The alternative we'll use involves defining the  $wp$  of an array assignment using an extended notion of substitution:

$wp(a[e_0] := e_1, p) \equiv [e_1/a[e_0]]p$  and  $\{[e_1/a[e_0]]p\} a[e_0] := e_1 \{p\}$

- Of course, we need to figure out what syntactic substitution for an array indexing expression means:  $[expression/a[e_0]](predicate)$
- Side note: The other way to handle array assignments, the Dijkstra / Gries technique, is to introduce a new kind of expression and view the array assignment  $a[e_0] := e_1$  as short for  $a :=$  this new kind of expression.

## D. Substitution for Array Elements

- We'll need to substitute into expressions and predicates. We'll tackle expressions first; below,  $a_1$  and  $a_2$  are different arrays.

- $[e/a_1[e_1]](a_2[e_2])$
- $[e/a_1[e_1]](a_1[e_2])$

- If  $a_1$  and  $a_2$  are different array names then substituting into  $a_2[e_2]$  will only require us to look at substituting into  $e_2$ :

$$[e/a_1[e_1]](a_2[e_2]) \equiv a_2[e_2'] \text{ where } e_2' \equiv [e/a_1[e_1]](e_2)$$

- When the the array names match, as in  $[e'/a[e]](a[k])$ , we have to check the indexes  $k$  and  $e$  for equality at runtime; to do that, we can use a conditional expression.
- Definition, case 1:**  $[e_1/a[e]](a[k]) \equiv (k = e ? e_1 : a[k])$ .
  - If  $k = e$  at runtime, then  $[e_1/a[e]](a[k]) = e_1$ .
  - If  $k \neq e$  at runtime, then  $[e_1/a[e]](a[k]) = a[k]$ .
- Example 1:**  $[5/a[0]](a[k]) \equiv (k = 0 ? 5 : a[k])$ .
- Example 2:**  $[e_1/a[j]](a[k]) \equiv (k = j ? e_1 : a[k])$ .
- Example 3:**  $[a[j]+1/a[j]](a[k]) \equiv (k = j ? a[j]+1 : a[k])$ .
  - Note: In  $[e_1/a[e_0]](a[k])$ , we don't substitute into  $e_1$ , even if it involves  $a$ .
- Example 4:**  $[a[i]/a[j]](a[k]) \equiv (k = j ? a[i] : a[k])$ .

## The General Case for Array Element Substitution

- When  $e_2$  is not just a simple variable or constant, then in  $[e/a[e_1]](a[e_2])$ , we have to check  $e_2$  for uses of  $a[\dots]$  and substitute for them also.
- Definition, case 2:**

$$[e_0/a[e_1]](a[e_2]) \equiv (e_2' = e_1 ? e_0 : a[e_2']) \text{ where } e_2' \equiv [e_0/a[e_1]](e_2).$$

- This subsumes the earlier case.

### Example 5

- Consider  $[5/a[0]](a[a[k]])$ — how should it behave? The nested  $a[k]$  should behave like 5 if  $k = 0$ , otherwise it's behaves like  $a[k]$  as usual. The outer  $a[\dots]$  should behave like 5 if its index behaves like 0, otherwise it should behave as  $a[\text{its index}]$ .
- Following the definition above, we get

$$[5/a[0]](a[a[k]]) \equiv (e_2' = 0 ? 5 : b[e_2'])$$

$$\text{where } e_2' \equiv [5/a[0]](a[k]) \equiv (k = 0 ? 5 : b[k])$$

- Substituting the (textual) value of  $e_2'$  gives us

$$\begin{aligned} & [5/a[0]](a[a[k]]) \\ & \equiv (k = 0 ? 5 : a[k]) = 0 ? 5 : a[k = 0 ? 5 : a[k]] \end{aligned}$$

- In the next section, we'll see how to simplify expressions so that we like this to get

$$k = 0 ? a[5] : (a[k] = 0 ? 5 : a[a[k]])$$

## E. Optimization of Static Cases

- Because  $[e_1/a[e_0]]e$  can result in a complicated piece of text, it can be useful to shorten it using various optimizations, similarly to how compilers can optimize code.
- All the optimizations below are intended to be done “statically” (at compile time) — we inspect the text of an expression before the code ever runs.
- For the easiest examples, if we know whether  $k = e_0$  or not, then we can optimize “ $k = e_0 ? \dots : \dots$ ” to just the true branch or the false branch.
- *General principle:*
  - If we know statically that  $k = e'$ , then we can optimize  $[e'/a[e]](a[k]) \equiv k = e ? e' : b[k]$  to  $e'$ .
  - If we know  $k \neq e'$ , then we can optimize to  $a[k]$ .
- *Notation:*  $e_1 \Rightarrow e_2$  (“ $e_1$  optimizes to  $e_2$ ”) means expression  $e_1$  can be replaced by expression  $e_2$ .
- *Example 6:*  $[e_1/a[2]](a[0]) \equiv 0 = 2 ? e_1 : a[0] \Rightarrow a[0]$ .
- *Example 7:*  $[e_1/a[2]](a[2]) \equiv 2 = 2 ? e_1 : a[2] \Rightarrow e_1$ .
- *Example 8:*  $[e'/a[x]](a[x]) \equiv x = x ? e' : b[x] \Rightarrow e'$ .

## F. Rules for Simplifying Conditional Expressions

- Let's identify some general rules for simplifying conditional expressions and predicates involving them. This will let us simplify calculation of  $wp$  for array assignments.
  - $T ? e_1 : e_2 \Rightarrow e_1$
  - $F ? e_1 : e_2 \Rightarrow e_2$
  - $e_0 ? e : e \Rightarrow e$
  - If  $(e_0 \rightarrow e_1 = e_2)$ , then  $e_0 ? e_1 : e_2 \Rightarrow e_2$
  - If  $(\neg e_0 \rightarrow e_1 = e_2)$ , then  $e_0 ? e_1 : e_2 \Rightarrow e_1$
- Let  $\Theta$  be a unary operator or relation and  $\oplus$  be a binary operation or relation
  - $\Theta(e ? e_1 : e_2) \Rightarrow e ? \Theta e_1 : \Theta e_2$
  - $(e ? e_1 : e_2) \oplus e_3 \Rightarrow e ? (e_1 \oplus e_3) \text{ else } (e_2 \oplus e_3)$
  - $b[e ? e_1 : e_2] \Rightarrow e ? b[e_1] : b[e_2]$
  - For any function  $f(\dots)$ ,  $f(e ? e_1 : e_2) \Rightarrow e ? f(e_1) : f(e_2)$

- If  $e$ ,  $e_1$ , and  $e_2$  are boolean expressions, then
  - $(e ? e_1 : F) \Leftrightarrow (e \wedge e_1)$
  - $(e ? F : e_2) \Leftrightarrow (\neg e \wedge e_2)$
  - $(e ? e_1 : T) \Leftrightarrow (e \rightarrow e_1) \Leftrightarrow (\neg e \vee e_1)$
  - $(e ? T : e_2) \Leftrightarrow (\neg e \rightarrow e_2) \Leftrightarrow (e \vee e_2)$
  - $(e ? e_1 : e_2) \Leftrightarrow ((e \rightarrow e_1) \wedge (\neg e \rightarrow e_2)) \Leftrightarrow ((e \wedge e_1) \vee (\neg e \wedge e_2)).$

- Example 9:

$$\begin{aligned}
 & wp(a[j] := a[j] + 1, a[k] < a[j]) \\
 & \equiv [a[j] + 1 / a[j]](a[k] < a[j]) \\
 & \equiv [a[j] + 1 / a[j]](a[k]) < [a[j] + 1 / a[j]](a[j]) \\
 & \equiv (k = j ? a[j] + 1 : a[k]) < a[j] + 1 \text{ (note we're optimizing here because } j = j) \\
 & \Leftrightarrow k = j ? (a[j] + 1 < a[j] + 1) : (a[k] < a[j] + 1) \\
 & \Leftrightarrow k = j ? F : a[k] < a[j] + 1 \\
 & \Leftrightarrow k \neq j \wedge a[k] < a[j] + 1
 \end{aligned}$$

This gives us the following correctness triple:

$$\{k \neq j \wedge a[k] < a[j] + 1\} a[j] := a[j] + 1 \{a[k] < a[j]\}$$

## G. Swapping Array Elements

- To illustrate the use of array references, let's look at the problem of swapping array elements.
- To swap simple variables  $x$  and  $y$  using a temporary variable  $u$ , we can use logical variables  $c$  and  $d$  and prove

$$\{x = c \wedge y = d\} u := x; x := y; y := u \{x = d \wedge y = c\}$$

- Example 10: For swapping  $a[m]$  and  $a[n]$ , we want to prove

$$\{a[m] = c \wedge a[n] = d\} u := a[m]; a[m] := a[n]; a[n] := u \{a[m] = d \wedge a[n] = c\}$$

As with simple variables, we can prove this holds by using  $wp$  to expand to the full proof outline. Let  $p \equiv a[m] = c \wedge a[n] = d$  and  $q \equiv a[m] = d \wedge a[n] = c$

$$\begin{aligned}
 & \{a[m] = c \wedge a[n] = d\} \\
 & \Rightarrow \{ [a[m]/u]((n = m ? u : a[n]) = d \wedge u = c) \} \\
 & = \{ (n = m ? a[m] : a[n]) = d \wedge a[m] = c \} \\
 & = \{ a[n] = d \wedge a[m] = c \} \\
 u := a[m]; & \{ [a[n]/a[m]]((n = m ? u : a[m]) = d \wedge u = c) \} \\
 & = \{ (n = m ? u : a[n]) = d \wedge u = c \} \\
 a[m] := a[n]; & \{ [u/a[n]](a[m] = d \wedge a[n] = c) \} \\
 & = \{ (n = m ? u : a[m]) = d \wedge u = c \} \\
 a[n] := u & \{ a[m] = d \wedge a[n] = c \}
 \end{aligned}$$