Array Element Assignments

CS 536: Science of Programming, Spring 2023

(Solved)

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

At the end of this work you should:

- Be able to perform textual substitution to replace an array element.
- Be able to calculate the wp of an array element assignment.

C. Questions

For each of the questions below, calculate the given weakest precondition. Then try logically simplifying it to something easier to read.

1. Calculate $wp(b[0] := 9,\ x > b[k])$.
2. Calculate $wp(b[k] := b[j],\ b[m] = 0)$.
3. Calculate $wp(b[k] := b[j],\ b[j] = z)$.
4. Calculate $wp(b[k] := 1,\ b[k] = b[j])$.
5. Calculate $wp(b[k] := x;\ b[j] := y,\ b[k] \neq b[j])$.
6. Calculate $wp(b[k] := x,\ b[b[k]] \neq b[k])$.
7. Is the triple $\{k < b[k] < b[j]\}\ b[b[k]] := b[j]\ \{b[k] \neq b[j]\}$ valid?
8. Define a predicate function $swapped(b_1, b_2, k, j)$ that yields true iff b_1 and b_2 are equal except their values at k and j are swapped.
9. Let the function $search(b, m, n, x) = k$ where if $m \le k \le n$ then $b[k] = x$ and if $k < m \vee k > n$, then no index for x exists in the range $m \dots n$. (I.e., for all j where $m \le j \le n$, we have $b[j] \neq x$.) Complete the predicate $\exists k\,.\,search(b, m, n, x) = k \rightarrow \dots$ where the ellipsis formalizes this description of what $search$ does.
10. In Example 5 in the notes, we had $(b[b[k]])[5 / b[0]]$

    $\equiv$ ***if*** (***if*** $k = 0$ ***then*** 5 ***else*** $b[k]$ ***fi***) $= 0$ ***then*** 5 ***else*** $b[$ ***if*** $k = 0$ ***then*** 5 ***else*** $b[k]$ ***fi*** $]$ ***fi***

    Show how to optimize this to ***if*** $k = 0$ ***then*** $b[5]$ ***else if*** $b[k] = 0$ ***then*** 5 ***else*** $b[b[k]]$ ***fi fi.***

Solution to Practice 21 (Array Element Assignments)

1. wp(b[0] := 9, x > b[k]) ≡ (x > b[k])[9/b[0]]
   ≡ x > ***if*** k = 0 ***then*** 9 ***else*** b[k] ***fi***
   ⇔ ***if*** k = 0 ***then*** x > 9 ***else*** x > b[k] ***fi***
   ⇔ (k = 0 ∧ x > 9) ∨ (k ≠ 0 ∧ x > b[k])

2. wp(b[k] := b[j], b[m] = 0) ≡ (b[m] = 0)[b[j]/b[k]]
   ≡ ***if*** m = k ***then*** b[j] ***else*** b[m] ***fi*** = 0
   ⇔ ***if*** m = k ***then*** b[j] = 0 ***else*** b[m] = 0 ***fi*** (one possible alternative rewriting)
   ⇔ (m = k → b[j] = 0) ∧ (m ≠ k → b[m] = 0) (another possible alternative rewriting)

3. wp(b[k] := b[j], b[j] = z) ≡ (b[j] = z)[b[j]/b[k]]
   ≡ ***if*** j = k ***then*** b[j] ***else*** b[j] ***fi*** = z
   ⇔ b[j] = z

4. wp(b[k] := 1, b[k] = b[j]) ≡ (b[k] = b[j])[1/b[k]]
   ≡ (b[k])[1/b[k]] = (b[j])[1/b[k]]
   ≡ 1 = (***if*** j = k ***then*** 1 ***else*** b[j] ***fi***)
   ⇔ k = j ∨ b[j] = 1 (⇔ k ≠ j → b[j] = 1 if you prefer → to ∨)

5. wp(b[k] := x; b[j] := y, b[k] ≠ b[j])
   ≡ wp(b[k] := x, wp(b[j] := y, b[k] ≠ b[j])

   For the embedded wp, wp(b[j] := y, b[k] ≠ b[j])
   ≡ (b[k] ≠ b[j])[y/b[j]]
   ≡ b[k] [y/b[j]] ≠ b[j] [y/b[j]]
   ≡ ***if*** k = j ***then*** y ***else*** b[k] ***fi*** ≠ y
   ⇔ k ≠ j ∧ b[k] ≠ y

   So wp(b[k] := x, wp(b[j] := y, b[k] ≠ b[j])
   ⇔ wp(b[k] := x, k ≠ j ∧ b[k] ≠ y)
   ≡ (k ≠ j ∧ b[k] ≠ y)[x/b[k]]
   ≡ k ≠ j ∧ b[k] [x/b[k]] ≠ y
   ≡ k ≠ j ∧ x ≠ y

   Intuitively, ***if*** k = j, ***then*** we can't have b[k] ≠ b[j]. Even ***if*** k ≠ j, we need x ≠ y to ensure that the assignments b[k] := x; b[j] := y make b[k] ≠ b[j].

6. For wp(b[k] := x, b[b[k]] ≠ b[k]), let's first look at substituting x for b[k] in b[b[k]]. It's complicated because we have to recursively substitute in the index of the outer b[…]. The general rule is

   $(b[e_2])[e_1/b[e_0]] \equiv (\textbf{\textit{if}}\ e_2' = e_0\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else}}\ b[e_2']\ \textbf{\textit{fi}})$ where $e_2' \equiv (e_2)[e_1/b[e_0]]$

So let $e' \equiv (b[k])[x/b[k]] \equiv x$, so

$(b[b[k]])[x/b[k]]$

$\equiv$ ***if*** $e' = k$ ***then*** x ***else*** $b[e']$ ***fi***

$\equiv$ ***if*** $x = k$ ***then*** x ***else*** $b[x]$ ***fi***

Then

$wp(b[k] := x, b[b[k]] \neq b[k])$

$\equiv (b[b[k]] \neq b[k])[x/b[k]]$

$\equiv (b[b[k]])[x/b[k]] \neq (b[k])[x/b[k]]$

$\equiv$ ***if*** $x = k$ ***then*** x ***else*** $b[x]$ ***fi*** $\neq x$

$\Leftrightarrow x \neq k \wedge b[x] \neq x$

7. For the triple to be valid, it's sufficient to show that its precondition implies the wp of the assignment and postcondition. I.e., $k < b[k] < b[j] \rightarrow wp(b[b[k]] := b[j], b[k] \neq b[j])$

   First let's calculate $wp(b[b[k]] := b[j], b[k] \neq b[j])$

   $\equiv (b[k] \neq b[j])[b[j]/b[b[k]]]$

   $\equiv (b[k])[b[j]/b[b[k]]] \neq (b[j])[b[j]/b[b[k]]]$

   $\equiv$ ***if*** $k = b[k]$ ***then*** $b[j]$ ***else*** $b[k]$ ***fi*** $\neq$ ***if*** $j = b[k]$ ***then*** $b[j]$ ***else*** $b[j]$ ***fi***

   $\Leftrightarrow$ ***if*** $k = b[k]$ ***then*** $b[j]$ ***else*** $b[k]$ ***fi*** $\neq b[j]$

   $\Leftrightarrow$ ***if*** $k = b[k]$ ***then*** $b[j] \neq b[j]$ ***else*** $b[k] \neq b[j]$ ***fi***

   $\Leftrightarrow k \neq b[k] \wedge b[k] \neq b[j]$

   Going back to our original question, if the implication below is valid, then our triple is valid (because we have the precondition of the triple implying the wp of its body and postcondition).

   $k < b[k] < b[j] \rightarrow wp(b[b[k]] := b[j], b[k] \neq b[j])$

   $\Leftrightarrow k < b[k] < b[j] \rightarrow k \neq b[k] \wedge b[k] \neq b[j]$

   $\Leftrightarrow T$

   So our original triple is indeed valid.

8. $swapped(b_1, b_2, k, j) \equiv b_1[k] = b_2[j] \wedge b_1[j] = b_2[k] \wedge (\forall m\,.\,(m \neq k \wedge m \neq j \rightarrow b_1[m] = b_2[m]))$

9. $\exists k\,.\,search(b, m, n, x) = k$

   $\rightarrow (m \leq k \leq n \rightarrow b[k] = x) \wedge (m > k \vee k > n \rightarrow (\forall j\,.\,m \leq j \leq n \rightarrow b[j] \neq x))$

   (Extra parentheses added for readability)

10. (Optimize $(b[b[k]])[5/b[0]]$)
    1. We start with ***if*** (***if*** $k = 0$ ***then*** 5 ***else*** $b[k]$ ***fi***) $= 0$ ***then*** 5 ***else*** $b[$ ***if*** $k = 0$ ***then*** 5 ***else*** $b[k]$ ***fi*** $]$ ***fi***

2. First, let's optimize the test for the outer ***if***:

   (***if*** k = 0 ***then*** 5 ***else*** b[k] ***fi***) = 0
   ↦ ***if*** k = 0 ***then*** 5 = 0 ***else*** b[k] = 0 ***fi***
   ↦ ***if*** k = 0 ***then*** F ***else*** b[k] = 0 ***f***
   ↦ k ≠ 0 ∧ b[k] = 0

3. Then let's optimize the false branch of the outer ***if***:

   b[***if*** k = 0 ***then*** 5 ***else*** b[k] ***fi***]
   ↦ ***if*** k = 0 ***then*** b[5] ***else*** b[b[k]] ***fi***

4. If we substitute (2) and (3) back into (1), we get

   if (***if*** k = 0 ***then*** 5 ***else*** b[k] ***fi***) = 0 ***then*** 5 ***else*** b[***if*** k = 0 ***then*** 5 ***else*** b[k] ***fi***] ***fi***
   ↦ ***if*** k ≠ 0 ∧ b[k] = 0 ***then*** 5 ***else if*** k = 0 ***then*** b[5] ***else*** b[b[k]] ***fi fi***

5. A little case analysis tells us that we want
   - b[5] when k = 0 (regardless of b[k] = or ≠ 0)
   - 5 when k ≠ 0 ∧ b[k] = 0
   - b[b[k]] when k ≠ 0 ∧ b[k] ≠ 0

6. This gives us

   if k = 0 ***then*** b[5]
   else /* k ≠ 0 */ ***if*** b[k] = 0 ***then*** 5
   else /* k ≠ 0 ∧ b[k] ≠ 0 */ b[b[k]] ***fi fi***