Array Element Assignments

CS 536: Science of Programming, Spring 2023

2023-04-03 pp. 2,4

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

- An array assignment $b[e_0] := e_1$ (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***: $\{T\}\ b[k] := b[j]\,;\ b[j] := b[j]+1\ \{b[k] < b[j]\}$
- The problem is what happens if $k = j$ at runtime: What is $wp(b[j] := b[j]+1,\ b[k] < b[j])$?
- The answer should be something like "If $k \neq j$ then $b[k] < b[j]+1$ else $b[j]+1 < b[j]+1$". (Note the else clause is false.)
- There are two alternatives for handling array assignments. The one we'll use involves defining the *wp* of an array assignment using an extended notion of textual substitution:

  $wp(b[e_0] := e_1,\ p) \equiv p[e_1 / b[e_0]]$ and $\{p[e_1 / b[e_0]]\}\ b[e_0] := e_1\ \{p\}$
- Of course, we need to figure out what syntactic substitution for an array indexing expression means: $(\mathit{predicate})[\mathit{expression} / b[e_0]]$
- 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 $b[e_0] := e_1$ as short for $b :=$ 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.

- If b and d are different arrays, then a substitution like $(b[m])[6 / d[2]]$ should simply $\equiv b[m]$. The situation can be more complicated: The substitution $(b[e])[6 / d[2]]$ has to recursively look for substitutions to do inside e.
  - $(b[e_2])[e_0 / d[e_1]] \equiv b[e_2']$ where $e_2' \equiv (e_2)[e_0 / d[e_1]]$. [2023-04-03]
- When the the array names match, as in $(b[k])[e_0 / b[e_1]]$, we have to check the indexes k and e_0 for equality at runtime; to do that, we can use a conditional expression.
- ***Definition (Substitution for an Array Element) — Simpler situation***
  - At runtime, if $k = e_1$, then $(b[k])[e_0 / b[e_1]] = e_0$. If $k \neq e_1$, then $(b[k])[e_0 / b[e_1]] = b[k]$. (The sense of "=" here is that the two expressions evaluate to the same value.)
    - Textually, $(b[k])[e_0 / b[e_1]] \equiv \textbf{if } k = e_1 \textbf{ then } e_0 \textbf{ else } b[k] \textbf{ fi}$.
- ***Example 1***: $(b[k])[5 / b[0]] \equiv (\textbf{if } k = 0 \textbf{ then } 5 \textbf{ else } b[k] \textbf{ fi})$.
- ***Example 2***: $(b[k])[e_0 / b[j]] \equiv (\textbf{if } k = j \textbf{ then } e_0 \textbf{ else } b[k] \textbf{ fi})$.
- ***Example 3***: $(b[k])[b[j]+1 / b[j]] \equiv (\textbf{if } k = j \textbf{ then } b[j]+1 \textbf{ else } b[k] \textbf{ fi})$.
  - Note: In $(b[k])[e_0 / b[e_1]]$, we don't substitute into e_0, even if it involves b.
- ***Example 4***: $(b[k])[b[i] / b[j]] \equiv (\textbf{if } k = j \textbf{ then } b[i] \textbf{ else } b[k] \textbf{ fi})$.

The General Case for Array Element Substitution

- When e_2 is not just a simple variable or constant, then in $(b[e_2])[e_0 / b[e_1]]$, we have to check e_2 for uses of $b[\ldots]$ and substitute for them also.
- ***Definition (Substitution for an Array Element) — General Case***

  $(b[e_2])[e_0 / b[e_1]] \equiv \textbf{if } e_2' = e_1 \textbf{ then } e_0 \textbf{ else } b[e_2'] \textbf{ fi}$ where $e_2' \equiv (e_2)[e_0 / b[e_1]]$.
- This subsumes the earlier case, since if $e_2 \equiv k$ then $e_2' \equiv k[e_0 / b[e_1]] \equiv k$. We get

$(b[k])[e_0 / b[e_1]] \equiv \textbf{if } k = e_1 \textbf{ then } e_0 \textbf{ else } b[k] \textbf{ fi}$

Example 5

- Consider $(b[b[k]])[5 / b[0]]$ — how should it behave? The inner, nested $b[k]$ should behave like 5 if $k = 0$, otherwise it should behaves like $b[k]$ as usual. The outer $b[\ldots]$ should behave like 5 if its index behaves like 0, otherwise it should behave as $b[\ldots]$.
- Following the definition above, we get

  $(b[b[k]])[5 / b[0]] \equiv \textbf{if } e_2' = 0 \textbf{ then } 5 \textbf{ else } b[e_2'] \textbf{ fi}$

  where $e_2' \equiv (b[k])[5 / b[0]] \equiv (\textbf{if } k = 0 \textbf{ then } 5 \textbf{ else } b[k] \textbf{ fi})$
- Substituting the (textual) value of e_2' gives us

$(b[b[k]])[5 / b[0]]$

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

$\textbf{then } 5$

$\textbf{else } b[\textbf{if } k = 0 \textbf{ then } 5 \textbf{ else } b[k] \textbf{ fi}] \textbf{ fi}$

- After optimization, this is equivalent to $\textbf{if } k = 0 \textbf{ then } b[5] \textbf{ else if } b[k] = 0 \textbf{ then } 5 \textbf{ else } b[b[k]] \textbf{ fi fi}$.

E. Optimization of Static Cases

- Because $e\,[\,e_0\,/\,b\,[\,e_1\,]\,]$ 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 or not $k = e_1$, the index of b we're looking for, then we can optimize ***if*** $k = e_0$ ***then*** e_1 ***else*** e_2 ***fi*** to just the true branch or the false branch.
- ***Notation:*** $e_1 \mapsto e_2$ ("e_1 optimizes to e_2") means we can replace expression e_1 with e_2.

General Principle (Static Optimizations)

- (Restricted case): For $(\,b\,[k]\,)\,[\,e_0\,/\,b\,[\,e_1\,]\,]$
  - If[1] $k = e_1$, then $(\,b\,[k]\,)\,[\,e_0\,/\,b\,[\,e_1\,]\,] \mapsto e_0$.
  - If $k \neq e_1$, then $(\,b\,[k]\,)\,[\,e_0\,/\,b\,[\,e_1\,]\,] \mapsto b\,[k]$.
- (General case): For $(\,b\,[\,e_2\,]\,)\,[\,e_0\,/\,b\,[\,e_1\,]\,]$, let $e_2{}' \equiv (\,e_2\,)\,[\,e_0\,/\,b\,[\,e_1\,]\,)$
  - If $e_2{}' = e_1$, then $(\,b\,[\,e_2\,]\,)\,[\,e_0\,/\,b\,[\,e_1\,]\,] \mapsto e_0$.
  - If $e_2{}' \neq e_1$, then $(\,b\,[\,e_2\,]\,)\,[\,e_0\,/\,b\,[\,e_1\,]\,] \mapsto b\,[k]$.

- ***Example 6***: $(\,b\,[0]\,)\,[\,e_1\,/\,b\,[2]\,] \equiv$ ***if*** $0 = 2$ ***then*** e_1 ***else*** $b\,[0]$ ***fi*** $\mapsto b\,[0]$.
- ***Example 7***: $(\,b\,[2]\,)\,[\,e_1\,/\,b\,[2]\,] \equiv$ ***if*** $2 = 2$ ***then*** e_1 ***else*** $b\,[2]$ ***fi*** $\mapsto e_1$.
- ***Example 8***:
  - $(\,b\,[0]\,)\,[\,e_0\,/\,b\,[1]\,] \equiv$ ***if*** $0 = 1$ ***then*** e_0 ***else*** $b\,[0]$ ***fi*** $\mapsto b\,[\,0\,]$.
  - $(\,b\,[1]\,)\,[\,e_0\,/\,b\,[1]\,] \equiv$ ***if*** $1 = 1$ ***then*** e_0 ***else*** $b\,[1]$ ***fi*** $\mapsto e_0$.
  - $(\,b\,[1]\,)\,[\,3/b\,[2]\,] \equiv$ ***if*** $1 = 2$ ***then*** 3 ***else*** $b\,[1]$ ***fi*** $\mapsto b[\,1\,]$.
  - $(\,b\,[x]\,)\,[\,e_0\,/\,b\,[x]\,] \equiv$ ***if*** $x = x$ ***then*** e_0 ***else*** $b\,[x]$ ***fi*** $\mapsto e_0$.

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.
  - (***if*** T ***then*** e_1 ***else*** e_2 ***fi***) $\mapsto e_1$.
  - (***if*** F ***then*** e_1 ***else*** e_2 ***fi***) $\mapsto e_2$.
  - (***if*** B ***then*** e ***else*** e ***fi***) $\mapsto e$.
  - If $(\,B \rightarrow e_1 = e_2\,)$, then (***if*** B ***then*** e_1 ***else*** e_2 ***fi***) $\mapsto e_2$.
  - If $(\,\neg\,B \rightarrow e_1 = e_2\,)$, then (***if*** B ***then*** e_1 ***else*** e_2 ***fi***) $\mapsto e_1$.

---

[1] The fuller version is "If we know that … then … ↦ …"

- Let ⊖ be a unary operator or relation and ⊕ be a binary operation or relation
  - $\ominus(\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else}}\ e_2\ \textbf{\textit{fi}}) \mapsto (\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ \ominus e_1\ \textbf{\textit{else}}\ \ominus e_2\ \textbf{\textit{fi}})$
  - $(\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else}}\ e_2\ \textbf{\textit{fi}}) \oplus e_3 \mapsto (\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ e_1 \oplus e_3\ \textbf{\textit{else}}\ e_2 \oplus e_3\ \textbf{\textit{fi}})$
  - $b[\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else}}\ e_2\ \textbf{\textit{fi}}] \mapsto \textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ b[e_1]\ \textbf{\textit{else}}\ b[e_2]\ \textbf{\textit{fi}}$
  - For any function $f(\dots)$, $f(\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else}}\ e_2\ \textbf{\textit{fi}}) \mapsto \textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ f(e_1)\ \textbf{\textit{else}}\ f(e_2)\ \textbf{\textit{fi}}$
- If B, B_1, and B_2 are boolean expressions, then
  - $(\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ B_1\ \textbf{\textit{else}}\ F\ \textbf{\textit{fi}}) \Leftrightarrow (B \wedge B_1)$
  - $(\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ F\ \textbf{\textit{else}}\ B_2\ \textbf{\textit{fi}}) \Leftrightarrow (\neg B \wedge B_2)$
  - $(\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ B_1\ \textbf{\textit{else}}\ T\ \textbf{\textit{fi}}) \Leftrightarrow (B \rightarrow B_1) \Leftrightarrow (\neg B \vee B_1)$
  - $(\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ T\ \textbf{\textit{else}}\ B_2\ \textbf{\textit{fi}}) \Leftrightarrow (\neg B \rightarrow B_2) \Leftrightarrow (B \vee B_2)$
  - $(\textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ B_1\ \textbf{\textit{else}}\ B_2\ \textbf{\textit{fi}}) \Leftrightarrow ((B \rightarrow B_1) \wedge (\neg B \rightarrow B_2)) \Leftrightarrow ((B \wedge B_1) \vee (\neg B \wedge B_2))$.
- We can also do reordering of ***if-else-if*** chains. E.g.,
  - $\textbf{\textit{if}}\ B_1\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else if}}\ B_2\ \textbf{\textit{then}}\ e_2\ \textbf{\textit{else}}\ e_3\ \textbf{\textit{fi}}$ evaluates e_1 if B_1 (regardless of B_2); it evaluates e_2[2023-04-03] if $\neg B_1 \wedge B_2$; and it evaluates e_3 if $\neg B_1 \wedge \neg B_2$.
  - So we (for example) swap e_2 and e_3 by changing the test slightly:
    - $\textbf{\textit{if}}\ B_1\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else if}}\ B_2\ \textbf{\textit{then}}\ e_2\ \textbf{\textit{else}}\ e_3\ \textbf{\textit{fi}}$

      $\mapsto \textbf{\textit{if}}\ B_1\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else if}}\ \neg B_2\ \textbf{\textit{then}}\ e_3\ \textbf{\textit{else}}\ e_2\ \textbf{\textit{fi}}$

      or

      $\mapsto \textbf{\textit{if}}\ \neg B_1 \wedge B_2\ \textbf{\textit{then}}\ e_2\ \textbf{\textit{else if}}\ B_1\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else}}\ e_3\ \textbf{\textit{fi}}$

      and so on.
- Similarly, we can move an inner ***if - else*** from the true branch of an outer ***if - else*** to the false branch of the outer ***if - else*** , in order to make an ***if-else-if*** chain. For example,
  - $\textbf{\textit{if}}\ B_1\ \textbf{\textit{then if}}\ B_2\ \textbf{\textit{then}}\ /*B_1 \wedge B_2*/\ e_1\ \textbf{\textit{else}}\ /*B_1 \wedge \neg B_2*/ e_2\ \textbf{\textit{fi else}}\ /*\neg B_1*/\ e_3\ \textbf{\textit{fi}}$

    $\mapsto \textbf{\textit{if}}\ \neg B_1\ \textbf{\textit{then}}\ e_3\ \textbf{\textit{else if}}\ B_2\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else}}\ e_2\ \textbf{\textit{fi}}\ \textbf{\textit{fi}}$

- ***Example 9***:

  $wp(b[j] := b[j]+1,\ b[k] < b[j])$

  $\equiv (b[k] < b[j])[b[j]+1 / b[j]]$

  $\equiv (b[k])[b[j]+1 / b[j]] < (b[j])[b[j]+1 / b[j]]$

  $\equiv \textbf{\textit{if}}\ k=j\ \textbf{\textit{then}}\ b[j]+1\ \textbf{\textit{else}}\ b[k]\ \textbf{\textit{fi}} < b[j]+1$

  $\Leftrightarrow \textbf{\textit{if}}\ k=j\ \textbf{\textit{then}}\ b[j]+1 < b[j]+1\ \textbf{\textit{else}}\ b[k] < b[j]+1\ \textbf{\textit{fi}}$

  $\Leftrightarrow \textbf{\textit{if}}\ k=j\ \textbf{\textit{then}}\ F\ \textbf{\textit{else}}\ b[k] < b[j]+1\ \textbf{\textit{fi}}$

  $\Leftrightarrow k \neq j \wedge b[k] < b[j]+1$

This gives us the following correctness triple:

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

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

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

- We can prove this program correct by expanding to a full proof outline; here we're using wp.

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

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

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

  As with simple variables, we can prove this holds by using wp to expand to the full proof outline.

  Let $p \equiv b[m] = c \wedge b[n] = d$ and $q \equiv b[m] = d \wedge b[n] = c$, then we can prove

  $\{p\}\ \{q_3\}\ u := b[m]\,;\ \{q_2\}\ b[m] := b[n]\,;\ \{q_1\}\ b[n] := u\ \{q\}$

  by using

  - $q_1 \equiv wp(b[n] := u, q) \equiv q[u/b[n]]$,
  - $q_2 \equiv wp(b[m] := b[n], q_1) \equiv q_1[b[n]/b[m]]$
  - $q_3 \equiv wp(u := b[m], q_2) \equiv q_2[b[m]/u]$
  - (and hopefully) $p \rightarrow q_3$

  We'll do this in steps.

  - $q_1 \equiv q[u/b[n]]$
    $\equiv (b[m] = d \wedge b[n] = c)[u/b[n]]$
    $\equiv (b[m] = d)[u/b[n]] \wedge (b[n] = c)[u/b[n]]$
    $\equiv (b[m])[u/b[n]] = d \wedge (b[n])[u/b[n]] = c$
    $\equiv (\textbf{if } m = n \textbf{ then } u \textbf{ else } b[m] \textbf{ fi}) = d \wedge u = c$ // Stop here for a purely syntactic result
  - $q_2 \equiv q_1[b[n]/b[m]]$
    $\equiv ((\textbf{if } m = n \textbf{ then } u \textbf{ else } b[m] \textbf{ fi}) = d \wedge u = c)[b[n]/b[m]]$
    $\equiv (\textbf{if } m = n \textbf{ then } u \textbf{ else } (b[m])[b[n]/b[m]] \textbf{ fi}) = d \wedge u = c$
    $\equiv (\textbf{if } m = n \textbf{ then } u \textbf{ else } b[n] \textbf{ fi}) = d \wedge u = c$
  - $q_3 \equiv q_2[b[m]/u]$
    $\equiv ((\textbf{if } m = n \textbf{ then } u \textbf{ else } b[n] \textbf{ fi}) = d \wedge u = c)[b[m]/u]$
    $\equiv (\textbf{if } m = n \textbf{ then } b[m] \textbf{ else } b[n] \textbf{ fi}) = d \wedge b[m] = c)$
    // Continuing with logical manipulation

$\Leftrightarrow (\textbf{if } m = n \textbf{ then } b[n] \textbf{ else } b[n] \textbf{ fi}) = d \wedge b[m] = c)$

// Because if $m = n$ then $b[m] = b[n]$

$\Leftrightarrow b[n] = d \wedge b[m] = c$.

- Since $p \equiv b[m] = c \wedge b[n] = d$, we get $p \rightarrow q_3$. (End of Example 10)