Array Element Assignments

CS 536: Science of Programming, Fall 2020

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 \( b[j]+1 < b[j]+1 \) (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 textual substitution:
    
    \[
    wp(b[e_0] := e_1, p) = p[e_1 / b[e_0]] \text{ and } \{p[e_1 / b[e_0]] b[e_0] := e_1 \{p\}}
    \]

- Of course, we need to figure out what syntactic substitution for an array indexing expression means: \((\text{predicate})[\text{expression}/b[e_0]]\)

- 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.

- If we have time later (we probably won't), we'll study this.
D. Substitution for Array Elements

- We'll need to substitute into expressions and predicates. We'll tackle expressions first; the new cases are
  - \((b_2[e_2])[e/b_1[e_1]]\) where \(b_1\) and \(b_2\) are different arrays
  - \((b_1[e_2])[e/b_1[e_1]]\)
- If \(b_1\) and \(b_2\) are different array names then substituting into \(b_2[e_2]\) will only require us to look at
  substituting into \(e_2\):
    \[(b_2[e_2])[e/b_1[e_1]] = b_2[e_2']\text{ where }e_2' = (e_2)[e/b_1[e_1]]\]
- When the the array names match, as in \((b[k])[e'/b[e]]\), 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:** \((b[k])[e_1/b[e]] = (if \ k = e \ then \ e_1 \ else \ b[k] \ fi)\).
- If \(k = e\) at runtime, then \((b[k])[e_1/b[e]]\) = \(e_1\).
- If \(k \neq e\) at runtime, then \((b[k])[e_1/b[e]]\) = \(b[k]\).

  **Example 1:** \((b[k])[5/b[0]] = (if \ k = 0 \ then \ 5 \ else \ b[k] \ fi)\).
  **Example 2:** \((b[k])[e_1/b[j]] = (if \ k = j \ then \ e_1 \ else \ b[k] \ fi)\).
  **Example 3:** \((b[k])[b[j]+1/b[j]] = (if \ k = j \ then \ b[j]+1 \ else \ b[k] \ fi)\).
- Note: In \((b[k])[e_1/b[e_0]]\), we don't substitute into \(e_1\), even if it involves \(b\).
  **Example 4:** \((b[k])[b[i]/b[j]] = (if \ k = j \ then \ b[i] \ else \ b[k] \ fi)\).

The general case for array element substitution

- More generally, for \((b[e_2])[e/b[e_1]]\), where \(e_2\) is not just a simple variable or constant, we have
  to check \(e_2\) for substitutions.

  **Definition, case 2:**
  \[(b[e_2])[e_0/b[e_1]] = (if \ e_2' = e_1 \ then \ e_0 \ else \ b[e_2'] \ fi)\text{ where }e_2' = (e_2)[e_0/b[e_1]].\]
- This subsumes the earlier case, since if \(e_2 = k\) then \(e_2' = k[e_0/b[e_1]] = k\). We get
  \[(b[k])[e/b[e_1]] = (if \ k = e_1 \ then \ e \ else \ b[k] \ fi)\]
  **Example 5:** Consider \((b[b[k]])[5/b[0]]\) — how should it behave? The nested \(b[k]\) should behave
  like 5 if \(k = 0\), otherwise it's behaves like \(b[k]\) as usual. The outer \(b[...]\) should behave like 5
  if its inner index behaves like 0, otherwise it should behave as \(b[its \ inner \ index]\).
- Following the definition above, we get
  \[(b[b[k]])[5/b[0]] = (if \ e_2' = 0 \ then \ 5 \ else \ b[e_2'] \ fi)\]
  where \(e_2' = (b[k])[5/b[0]] = (if \ k = 0 \ then \ 5 \ else \ b[k] \ fi)\)
- Substituting the (textual) value of \(e_2'\) gives us
  \[(b[b[k]])[5/b[0]] = if \ (if \ k = 0 \ then \ 5 \ else \ b[k] \ fi) = 0\]
Let's identify some general rules for simplifying conditional expressions and predicates involving

Example 8
So let's use the following optimizations:

Example 6: We can optimize \( b[0]|e_1/b[2]| = if 0 = 2 then e_1 else b[0] fi \) to just \( b[0] \).

Example 7: Similarly, we can optimize \( b[2]|e_1/b[2]| = if 2 = 2 then e_1 else b[2] fi \) to just \( e_1 \).

So let's use the following optimizations:

Example 8:
- For \( b[0]|e'/b[1]| \), use \( b[0] \), not \( if 0 = 1 then e' else b[0] fi \).
- For \( b[1]|e'/b[1]| \), use \( e' \), not \( if 1 = 1 then e' else b[1] fi \).
- For \( b[x]|e'/b[x]| \), use \( e' \), not \( if x = x then e' else b[x] fi \).

F. Rules for Simplifying Conditional Expressions

Let's identify some general rules for simplifying conditional expressions and predicates involving

- \( if \ T \ then \ e_1 \ else \ e_2 \ fi = e_1 \) and \( if \ F \ then \ e_1 \ else \ e_2 \ fi = e_2 \)
- \( if \ B \ then \ e \ else \ e \ fi \) = e

\( if \ (B \rightarrow e_1 = e_2) \), then \( if \ B \ then \ e_1 \ else \ e_2 \ fi = e_2 \)

\( if \ (\neg B \rightarrow e_1 = e_2) \), then \( if \ B \ then \ e_1 \ else \ e_2 \ fi = e_1 \)

Let \( \odot \) be a unary operator or relation and \( \oplus \) be a binary operation or relation

- \( \odot \ (if \ B \ then \ e_1 \ else \ e_2 \ fi) = (if \ B \ then \ \odot \ e_1 \ else \ \odot \ e_2 \ fi) \)
- \( (if \ B \ then \ e_1 \ else \ e_2 \ fi) \oplus e_3 = (if \ B \ then \ e_1 \oplus e_3 \ else \ e_2 \oplus e_3 \ fi) \)
- \( b[\ if \ B \ then \ e_1 \ else \ e_2 \ fi \] = if \ B \ then \ b[e_1] else \ b[e_2] fi \)
- For any function \( f(...) \), \( f(if \ B \ then \ e_1 \ else \ e_2 \ fi) = if \ B \ then \ f(e_1) else \ f(e_2) \ fi \)

If \( B, B_1, \) and \( B_2 \) are boolean expressions, then

- \( if \ B \ then \ B_1 \ else \ F \ fi \) \( \equiv \ (B \land B_1) \)
- \( if \ B \ then \ F \ else \ B_2 \ fi \) \( \equiv \ (\neg B \land B_2) \)
• \((\text{if } B \text{ then } B_1 \text{ else } T \text{ fi}) \iff (B \rightarrow B_1) \iff (\neg B \vee B_1)\)
• \((\text{if } B \text{ then } T \text{ else } B_2 \text{ fi}) \iff (\neg B \rightarrow B_2) \iff (B \vee B_2)\)
• \((\text{if } B \text{ then } B_1 \text{ else } B_2 \text{ fi}) \iff ((B \rightarrow B_1) \land (\neg B \rightarrow B_2)) \iff ((B \land B_1) \lor (\neg B \land B_2)).\)

**Example 9:**

\[
\begin{align*}
\text{wp}(b[j]):= b[j]+1, \: b[k] &< b[j]) \\
&= (b[k]<b[j][b[j]+1/b[j]]) \\
&= (b[k]) [b[j]+1/b[j]] < (b[j]) [b[j]+1/b[j]] \\
&= \text{if } k = j \text{ then } b[j]+1 \text{ else } b[k] \text{ fi } < b[j]+1 \\
&\iff \text{if } k = j \text{ then } b[j]+1 < b[j]+1 \text{ else } b[k] < b[j]+1 \text{ fi} \\
&\iff \text{if } k = j \text{ then } F \text{ else } b[k] < b[j]+1 \text{ fi} \\
&\iff k \neq j \land b[k] < b[j]+1
\end{align*}
\]

*So we know \(k \neq j \land b[k] < b[j]+1\) \(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

\[
\begin{align*}
\{x=c \land y=d\} & u:= x; \: x:= y; \: y:= u \{x=d \land y=c\}
\end{align*}
\]
• We can prove this program correct by expanding to a full proof outline; here we’re using \(\text{wp}\).

\[
\begin{align*}
\{x=c \land y=d\} & u:= x; \: x:= y; \: y:= u \{x=d \land y=c\}
\end{align*}
\]

**Example 10:** For swapping \(b[m]\) and \(b[n]\), we want to prove

\[
\begin{align*}
\{b[m]=c \land b[n]=d\} & u:= b[m]; \: b[m]:= b[n]; \: b[n]:= u \{b[m]=d \land b[n]=c\}
\end{align*}
\]

As with simple variables, we can prove this holds by using \(\text{wp}\) to expand to the full proof outline

\[
\begin{align*}
\{p\} & \{q_3\} u:= b[m]; \: \{q_2\} b[m]:= b[n]; \: \{q_1\} b[n]:= u \{q\}
\end{align*}
\]

where

• \(p = b[m]=c \land b[n]=d\)
• \(q = b[m]=d \land b[n]=c\)
• \(q_1 = \text{wp}(b[n]:= u, q) = q[u/b[n]],\)
• \(q_2 = \text{wp}(b[m]:= b[n], q_1) = q_1[b[n]/b[m]]\)
• \(q_3 = \text{wp}(u:= b[m], q_2) = q_2[b[m]/u]\)
• and (we hope), \(p \rightarrow q_3\).

We’ll do this in steps.

\[
q_1 = q[u/b[n]] = (b[m]=d \land b[n]=c) [u/b[n]]
\]
\[(b[m] = d)[u/b[n]] \land (b[n] = c)[u/b[m]]\]
\[(b[m])[u/b[n]] = d \land (b[n])[u/b[m]] = c\]
\[= (\text{if } m = n \quad \text{then } u \quad \text{else } b(m) \quad \text{fi}) = d \land u = c \quad \text{// Stop here for a purely syntactic result}\]

\[q_2 = q_1[b[n]/b[m]]\]
\[= (\text{if } m = n \quad \text{then } u \quad \text{else } b[m] \quad \text{fi}) = d \land u = c \quad \text{[b[n]/b[m]]}\]
\[= (\text{if } m = n \quad \text{then } u \quad \text{else } (b[m])[b[n]/b[m]] \quad \text{fi}) = d \land u = c\]
\[= (\text{if } m = n \quad \text{then } u \quad \text{else } b[n] \quad \text{fi}) = d \land u = c\]

\[q_3 = q_2[b[m]/u]\]
\[= (\text{if } m = n \quad \text{then } u \quad \text{else } b[n] \quad \text{fi}) = d \land u = c \quad \text{[b[m]/u]}\]
\[= (\text{if } m = n \quad \text{then } b[m] \quad \text{else } b[n] \quad \text{fi}) = d \land b[m] = c\]
\[= \quad \text{// Continuing with logical manipulation}\]
\[\leftrightarrow (\text{if } m = n \quad \text{then } b[n] \quad \text{else } b[n] \quad \text{fi}) = d \land b[m] = c \quad \text{// if } m = n \quad \text{then } b[m] = b[n]\]
\[\leftrightarrow b[n] = d \land b[m] = c\]

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