Substitution in Arrays

- 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. For example, consider the following question.
  - Assuming $b[j]$ and $b[k]$ are safe, then what is $wp(b[j] \coloneqq b\ [j] + 1, b[k] < b[j])$? Some might say it is $b[k] < b[j] + 1$ but what if $k = j$? In that case, will the wp be $b[j] + 1 < b[j] + 1 \Leftrightarrow F$?

  The problem comes from the ambiguity of the meaning of $(e)\ [e_1 \ / \ b[e_0]]$. Let's use an example to understand the syntactic substitution in an array.

1. What is $(b[m])[6 \ / \ d[2]]$?
   a) Let m be a named constant.
   - If b and d are different array, then this is simple: there will be no expression $d[2]$ in array $b[m]$ then $(b[m])[6 \ / \ d[2]] = b[m]$.
   - If b and d are the same array, then we need to consider whether $m = 2$: if $m = 2$ then $(b[m])[6 \ / \ d[2]] = 6$ or else it $= b[m]$. Here we are analyzing the values, so we use "=".

   b) How about $(b[e])[6 \ / \ d[2]]$ where e is an expression and $b \not\equiv d$?
   - We need to look recursively into e since array d might appear in e; then, $(b[e])[6 \ / \ d[2]] = b\left[e[6 \ / \ d[2]]\right]$.

   c) How about $(b[e])[6 \ / \ d[2]]$ where e is an expression and $b \equiv d$?
   - If e is evaluated to 2 at run time, then $(b[e])[6 \ / \ d[2]] = 6$, or else we will look recursively into e like the case in b).

- Here we give the definition of substitution in arrays:
  $(b[e_2])\ [e_1 \ / \ b[e_0]] \equiv$ **if** $e_2' = e_0$ **then** e_1 **else** $b[e_2']$ **fi**, where $e_2' \equiv (e_2)[e_1 \ / \ b[e_0]]$.

  - This definition covers all cases in Example 1 while the substitution happens in the same array:
    - When e_2 is a named constant, aka $e_2 \equiv k$, then we get $e_2' \equiv k[e_1 / \ b[e_0]] \equiv k$, and then $(b[k])\ [e_1 \ / \ b[e_0]] \equiv$ **if** $k = e_0$ **then** e_1 **else** $b[k]$ **fi**.

2. Finish the following syntactic substitutions.
   a) $(b[k])[5 \ / \ b[0]] \equiv$ **if** $k = 0$ **then** 5 **else** $b[k]$ **fi**
   b) $(b[k])[e_0 \ / \ b[j]] \equiv$ **if** $k = j$ **then** e_0 **else** $b[k]$ **fi**
   c) $(b[k])[b[j] + 1 \ / \ b[j]] \equiv$ **if** $k = j$ **then** $b[j] + 1$ **else** $b[k]$ **fi**
   Note that, we will keep e_1 (in this case $b[j] + 1$) as it is, even if it involves b.

   d) $(b[b[k]])[5 \ / \ b[0]]$
   The inner $b[k]$ need to be take care first, and $(b[k])[5 \ / \ b[0]] \equiv$ **if** $k = 0$ **then** 5 **else** $b[k]$ **fi**. Then,
   $(b[b[k]])[5 \ / \ b[0]]$
   $\equiv$ **if** $(\text{if } k = 0 \text{ then } 5 \text{ else } b[k] \text{ fi}) = 0$ **then** 5 **else** $b[\text{if } k = 0 \text{ then } 5 \text{ else } b[k] \text{ fi}]$ **fi**
   $\mapsto$ **if** $k = 0$ **then** $b[5]$ **else** **if** $b[k] = 0$ **then** 5 **else** $b[b[k]]$ **fi** **fi**

- In Example 2.d) we simplified a complicated expression to a simpler expression. Formally, we call this operation **optimization**, it means we replace an expression with a (shorter) expression that is *semantically equal*. It is written as $e_1 \mapsto e_2$ ("e_1 optimizes to e_2").
  - We introduce this definition here because syntactic substitutions in arrays usually end up with conditional expressions or, even worse, nested conditional statements. So, we point out that in our programming language we can optimize code to make the execution easier.
  - The optimization we talk about here is *static*. This is like how compiler optimizes code; the optimization is done before the code runs.
  - Since to $e_1 \mapsto e_2$ we need $e_1 \Leftrightarrow e_2$, it is okay to just use "$\Leftrightarrow$" to represent optimization. But "$e_1 \Leftrightarrow e_2$" emphasizes that e_1 and e_2 are semantically equal, and "$e_1 \mapsto e_2$" emphasizes that e_1 is (or can be) optimized to e_2 while we compile the code.

- The general optimization principle is simple. For $(b[e_2])\,[e_1 \,/\, b[e_0]]$, let $e_2' \equiv (e_2)[e_1 / b[e_0]]$:
  - If $e_2' = e_0$, then $(b[e_2])\,[e_1 \,/\, b[e_0]] \mapsto e_1$.
  - If $e_2' \neq e_0$, then $(b[e_2])\,[e_1 \,/\, b[e_0]] \mapsto b[e'_2]$.

3. Optimize the following expressions.
   a) $(b[0])[e_1 \,/\, b[2]] \equiv \textbf{if } 0 = 2 \textbf{ then } e_1 \textbf{ else } b[0] \textbf{ fi} \mapsto b[0]$
   b) $(b[1])[e_1 \,/\, b[1]] \equiv \textbf{if } 1 = 1 \textbf{ then } e_1 \textbf{ else } b[0] \textbf{ fi} \mapsto e_1$

Rules for Optimizing Condition Expressions

Let's identify some general rules for optimizing conditional expressions and predicates involving them. This will let us simplify calculation of wlp or wp for array assignments.

- $(\textbf{if } T \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}) \mapsto e_1$
- $(\textbf{if } F \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}) \mapsto e_2$
- $(\textbf{if } B \textbf{ then } e \textbf{ else } e \textbf{ fi}) \mapsto e$
- If we know that $B \Rightarrow e_1 = e_2$, then $(\textbf{if } B \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}) \mapsto e_2$.
  - Since B can imply that $e_1 = e_2$, then no matter whether B is true or not, we always have e_2.
- If we know that $\neg B \Rightarrow e_1 = e_2$, then $(\textbf{if } B \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}) \mapsto e_1$.

Let op_1 be a unary operator, such as "$\neg$"…; and op_2 be a binary operator such as "$+$", "$<$"…

- $op_1\ (\textbf{if } B \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}) \mapsto \textbf{if } B \textbf{ then } op_1(e_1) \textbf{ else } op_1(e_2) \textbf{ fi}$
- $(\textbf{if } B \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi})\ op_2\ e_3 \mapsto \textbf{ if } B \textbf{ then } e_1\ op_2\ e_3 \textbf{ else } e_2\ op_2\ e_3 \textbf{ fi}$
- $b[\ \textbf{if } B \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}\ ] \mapsto \textbf{if } B \textbf{ then } b[e_1] \textbf{ else } b[e_2] \textbf{ fi}$
- $f(\textbf{if } B \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}) \mapsto \textbf{if } B \textbf{ then } f(e_1) \textbf{ else } f(e_2) \textbf{ fi}$

Let B, B_1, B_2 be Boolean expression.

- $(\textbf{if } B \textbf{ then } B_1 \textbf{ else } B_2 \textbf{ fi}) \mapsto (B \wedge B_1) \vee (\neg B \wedge B_2)$
- $(\textbf{if } B \textbf{ then } B_1 \textbf{ else } B_2 \textbf{ fi}) \mapsto (B \rightarrow B_1) \wedge (\neg B \rightarrow B_2)$
  - There are two ways to understand a condition expression.
    - If we have B, then we also have B_1; if we have $\neg B$, then we also have B_2. Since we can have either B or $\neg B$, so we have either $(B \wedge B_1)$ or $(\neg B \wedge B_2)$. Thus, $(\textbf{if } B \textbf{ then } B_1 \textbf{ else } F \textbf{ fi}) \Leftrightarrow (B \wedge B_1) \vee (\neg B \wedge B_2)$.
    - B implies B_1 and $\neg B$ implies B_2; thus, $(\textbf{if } B \textbf{ then } B_1 \textbf{ else } F \textbf{ fi}) \Leftrightarrow (B \rightarrow B_1) \wedge (\neg B \rightarrow B_2)$.

- $(\textbf{if } B \textbf{ then } B_1 \textbf{ else } F \textbf{ fi}) \mapsto (B \wedge B_1)$

- $(\textbf{if } B \textbf{ then } B_1 \textbf{ else } F \textbf{ fi}) \Leftrightarrow (B \wedge B_1) \vee (\neg B \wedge F) \Leftrightarrow (B \wedge B_1)$.
- $(\textbf{if } B \textbf{ then } F \textbf{ else } B_2 \textbf{ fi}) \mapsto (\neg B \wedge B_2)$
- $(\textbf{if } B \textbf{ then } B_1 \textbf{ else } T \textbf{ fi}) \mapsto (B \rightarrow B_1)$
- $(\textbf{if } B \textbf{ then } B_1 \textbf{ else } T \textbf{ fi}) \mapsto (\neg B \vee B_1)$
  - $(\textbf{if } B \textbf{ then } B_1 \textbf{ else } T \textbf{ fi}) \Leftrightarrow (B \rightarrow B_1) \wedge (\neg B \rightarrow T) \Leftrightarrow (B \rightarrow B_1)$

- $(\textbf{if } B \textbf{ then } T \textbf{ else } B_2 \textbf{ fi}) \mapsto (\neg B \rightarrow B_2)$
- $(\textbf{if } B \textbf{ then } T \textbf{ else } B_2 \textbf{ fi}) \mapsto (B \vee B_2)$

Remind that, we have chain conditional expressions: $\textbf{if } B_1 \textbf{ then } e_1 \textbf{ else } B_2 \textbf{ then } e_2 \textbf{ else } e_3 \textbf{ fi}$, syntactically it is equivalent to:

$\textbf{if } B_1 \textbf{ then}$
$\quad e_1$
$\textbf{else}$
$\quad \textbf{if } B_2 \textbf{ then } e_2 \textbf{ else } e_3 \textbf{ fi}$
$\textbf{fi}$

For a chain $\textbf{if} - \textbf{else} - \textbf{if}$ expression, we can change the order of branches, if we carefully consider the condition on each branch.

- $\textbf{if } B_1 \textbf{ then } e_1 \textbf{ else } B_2 \textbf{ then } e_2 \textbf{ else } e_3 \textbf{ fi}$ $\quad \#\ B_1 \rightarrow e_1, \neg B_1 \wedge B_2 \rightarrow e_2, \neg B_1 \wedge \neg B_2 \rightarrow e_3$
  $\Leftrightarrow \textbf{if } B_1 \textbf{ then } e_1 \textbf{ else } \neg B_2 \textbf{ then } e_3 \textbf{ else } e_2 \textbf{ fi}$
  $\Leftrightarrow \textbf{if } \neg B_1 \wedge B_2 \textbf{ then } e_2 \textbf{ else } B_1 \textbf{ then } e_1 \textbf{ else } e_3 \textbf{ fi}$
  $\#\ \neg B_1 \wedge B_2 \rightarrow e_2, (B_1 \vee \neg B_2) \wedge B_1 \Leftrightarrow B_1 \rightarrow e_1, (B_1 \vee \neg B_2) \wedge \neg B_1 \Leftrightarrow \neg B_1 \wedge \neg B_2 \rightarrow e_3$

  $\Leftrightarrow \textbf{if } \neg B_1 \wedge B_2 \textbf{ then } e_2 \textbf{ else } \neg B_1 \textbf{ then } e_3 \textbf{ else } e_1 \textbf{ fi}$
  $\Leftrightarrow \textbf{if } \neg B_1 \wedge \neg B_2 \textbf{ then } e_3 \textbf{ else } B_1 \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}$
  $\#\ \neg B_1 \wedge \neg B_2 \rightarrow e_3, (B_1 \vee B_2) \wedge B_1 \Leftrightarrow B_1 \rightarrow e_1, (B_1 \vee B_2) \wedge \neg B_1 \Leftrightarrow \neg B_1 \wedge B_2 \rightarrow e_2$

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

Now, let's go back to the first question of the class.

4. Let j and k be two named constants that are at least 0 and less than $size(b)$. Calculate $wp(b[j] \coloneqq b\,[j] + 1, b[k] < b[j])$.

   $wp(b[j] \coloneqq b\,[j] + 1, b[k] < b[j])$
   $\equiv (b[k] < b[j])\,[b\,[j] + 1 \,/\, b\,[j]]$
   $\equiv (b[k])\,[b\,[j] + 1 \,/\, b\,[j]] < (b[j])\,[b\,[j] + 1 \,/\, b\,[j]]$
   $\equiv (\textbf{if } k = j \textbf{ then } b\,[j] + 1 \textbf{ else } b[k] \textbf{ fi}) < (b[j] + 1)$
   $\mapsto \textbf{if } k = j \textbf{ then } (b\,[j] + 1) < (b[j] + 1) \textbf{ else } b[k] < (b[j] + 1) \textbf{ fi}$
   $\mapsto \textbf{if } k = j \textbf{ then } F \textbf{ else } b[k] < (b[j] + 1) \textbf{ fi}$
   $\mapsto k \neq j \wedge b[k] < (b[j] + 1)$

   - This gives us a valid triple $\{k \neq j \wedge b[k] < (b[j] + 1)\}\ b[j] \coloneqq b\,[j] + 1\ \{b[k] < b[j]\}$

5. Correct a full proof outline of a program that swaps the values of primitive-type variables x and y.

- To swaps the values of x and y, we need the help of a temporary variable u, then we can create the following minimal proof outline:

$$\{x = x_0 \wedge y = y_0\}\, u \coloneqq x; x \coloneqq y; y \coloneqq u\, \{x = y_0 \wedge y = x_0\}$$

- We can keep using backward assignments to create the following full proof outline:

$\{x = x_0 \wedge y = y_0\}\, u \coloneqq x; \{y = y_0 \wedge u = x_0\}\, x \coloneqq y; \{x = y_0 \wedge u = x_0\}\, y \coloneqq u\, \{x = y_0 \wedge y = x_0\}$

6. Create a full proof outline of a program that swaps $b[m]$ and $b[n]$, assuming that m and n are natural numbers less than $size(b)$.
   - Like question 5, we need to prove the following minimal proof outline:

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

   - If we keep using backward assignments, then we can come up with the following full proof outline:

$$\{b[m] = c \wedge b[n] = d\}\, \{q_0\}\, u \coloneqq b[m]; \{q_1\}\, b[m] \coloneqq b[n]; \{q_2\}\, b[n] \coloneqq u\, \{b[m] = d \wedge b[n] = c\}$$

   Let's calculate q_2, q_1 and q_0. Note that, we also need to prove $b[m] = c \wedge b[n] = d \Rightarrow q_0$.

$$\begin{aligned}
q_2 &\equiv (b[m] = d \wedge b[n] = c)\,[u \,/\, b[n]] \\
&\equiv (b[m] = d)\,[u \,/\, b[n]] \wedge (b[n] = c)\,[u \,/\, b[n]] \\
&\equiv (b[m])\,[u \,/\, b[n]] = d \wedge (u = c) \\
&\equiv (\textbf{if } m = n \textbf{ then } u \textbf{ else } b[m] \textbf{ fi}) = d \wedge (u = c) \quad \text{\# Stop here for pure syntactic result}
\end{aligned}$$

$$\begin{aligned}
q_1 &\equiv \big((\textbf{if } m = n \textbf{ then } u \textbf{ else } b[m] \textbf{ fi}) = d \wedge (u = c)\big)\,[b[n] \,/\, b[m]] \\
&\equiv (\textbf{if } m = n \textbf{ then } u \textbf{ else } b[m] \textbf{ fi})[b[n] \,/\, b[m]] = d \wedge (u = c) \\
&\equiv (\textbf{if } m = n \textbf{ then } u \textbf{ else } b[n] \textbf{ fi}) = d \wedge (u = c)
\end{aligned}$$

$$\begin{aligned}
q_0 &\equiv \big((\textbf{if } m = n \textbf{ then } u \textbf{ else } b[n] \textbf{ fi}) = d \wedge (u = c)\big)\,[b[m] \,/\, u] \\
&\equiv (\textbf{if } m = n \textbf{ then } b[m] \textbf{ else } b[n] \textbf{ fi}) = d \wedge (b[m] = c) \\
&\qquad \text{\# Let's try to optimize } q_0. \\
&\qquad \text{\# } m = n \text{ implies } b[m] = b[n] \\
&\mapsto b[n] = d \wedge (b[m] = c)
\end{aligned}$$

   It is obvious that the precondition $b[m] = c \wedge b[n] = d \Leftrightarrow q_0$ so the proof is done. Here is the full proof outline:

$$\begin{aligned}
&\{b[m] = c \wedge b[n] = d\}\, u \coloneqq b[m]; \\
&\{(\textbf{if } m = n \textbf{ then } u \textbf{ else } b[n] \textbf{ fi}) = d \wedge (u = c)\}\, b[m] \coloneqq b[n]; \\
&\{(\textbf{if } m = n \textbf{ then } u \textbf{ else } b[m] \textbf{ fi}) = d \wedge (u = c)\}\, b[n] \coloneqq u\, \{b[m] = d \wedge b[n] = c\}
\end{aligned}$$