More Examples of Creating Loops

1. Given two non-empty sorted arrays b_1 and b_2, find the least indices i and j such that $b_1[i] = b_2[j]$; if no such i and j exist, end with $i = n$ or $j = m$ such that $n = size(b_1)$ and $m = size(b_2)$.

   - We have seen the three-array version of this problem when we introduce nondeterministic statements: our algorithm starts with $i = j = 0$ then increase either i or j in each iteration. This time we focus on the loop invariant and termination.

   - Let's start with the bound expression this time. Since we are increase i and j, so we define bound function $t(i,j) \equiv (n-i) + (m-j)$. It is easy to see that $t(i,j) \geq 0$.

   - The precondition is also easy to see: $p_0 \equiv size(b_1) = n > 0 \wedge size(b_2) = m > 0 \wedge Sorted(b_1) \wedge Sorted(b_2)$, where $Sorted(b) \equiv \forall 0 \leq k < size(b) - 1.\, b[k] \leq b[k+1]$. Since $Sorted(b_1) \wedge Sorted(b_2)$ is always true during the program and our searching procedure doesn't change it; we will only show it in the precondition and omit it everywhere else.

   - While discussing the three-array version, we mentioned that our algorithm will only return the left-most match. So, if the program ends with $b_1[i]$ and $b_2[j]$, then there is no match on the left of i and j, no matter whether at least one of i, j equals n or not. This is similar to the postcondition of the linear search, we can write postcondition $q \equiv InRange(i,j) \wedge NoMatch(i,j) \wedge (i < n \wedge j < m \rightarrow b_1[i] = b_2[j])$, where $InRange(i,j) \equiv 0 \leq i \leq n \wedge 0 \leq j \leq m$ and $NoMatch(i,j) \equiv \forall 0 \leq i' < i.\, \forall 0 \leq j' < j.\, b_1[i'] \neq b_2[j']$.

   - Like linear search, we can also get invariant by dropping of the last conjunct, and $p \equiv InRange(i,j) \wedge NoMatch(i,j)$, then we have loop condition $B \equiv \neg(i < n \wedge j < m \rightarrow b_1[i] = b_2[j]) \Leftrightarrow i < n \wedge j < m \wedge b_1[i] \neq b_2[j]$. Then we can get the following partial program:

     $\{size(b_1) = n > 0 \wedge size(b_2) = m > 0 \wedge Sorted(b_1) \wedge Sorted(b_2)\}$
     $\{InRange(0,0) \wedge NoMatch(0,0)\}$
     $i \coloneqq 0; j \coloneqq 0;$
     $\{\textbf{inv}\ p \equiv InRange(i,j) \wedge NoMatch(i,j)\}\ \{\textbf{bd}\ t(i,j) \equiv (n-i) + (m-j)\}$
     while $B \equiv i < n \wedge j < m \wedge b_1[i] \neq b_2[j]$ **do**
     $\{p \wedge B \wedge t(i,j) = t_0\}$
     … increase i or j, and maybe something else …
     $\{p \wedge t(i,j) < t_0\}$
     od
     $\{p \wedge (i < n \wedge j < m \rightarrow b_1[i] = b_2[j])\}$
     $\{q \equiv InRange(i,j) \wedge NoMatch(i,j) \wedge (i < n \wedge j < m \rightarrow b_1[i] = b_2[j])\}$

   - What should be included in the loop body? Let's use a deterministic program this time. We want to increase $i \coloneqq i + 1$ when $b_1[i] < b_2[j]$, since $b_2[j]$ is already too large so increasing j won't help; symmetrically, we want to increase $j \coloneqq j + 1$ when $b_1[i] > b_2[j]$. With a conditional statement, we have the following partial proof outline:

     $\{size(b_1) = n > 0 \wedge size(b_2) = m > 0 \wedge Sorted(b_1) \wedge Sorted(b_2)\}\ \{InRange(0,0) \wedge NoMatch(0,0)\}$
     $i \coloneqq 0; j \coloneqq 0;$

$\{\textbf{inv}\ p \equiv InRange(i,j) \wedge NoMatch(i,j)\}\ \{\textbf{bd}\ t(i,j) \equiv (n-i)+(m-j)\}$
while $B \equiv i < n \wedge j < m \wedge b_1[i] \neq b_2[j]$ **do**
$\{p \wedge B \wedge t(i,j) = t_0\}$
if $b_1[i] < b_2[j]$ **then** # Conditional Rule 1
$\{p \wedge B \wedge t(i,j) = t_0 \wedge b_1[i] < b_2[j]\}$
$\{p[i+1\ /\ i] \wedge t(i+1,j) < t_0\}\ i \coloneqq i+1\ \{p \wedge t(i,j) < t_0\}$
else $\#b_1[i] > b_2[j]$ since $b_1[i] \neq b_2[j]$
$\{p \wedge B \wedge t(i,j) = t_0 \wedge b_1[i] > b_2[j]\}$
$\{p[j+1\ /\ j] \wedge t(i,j+1) < t_0\}\ j \coloneqq j+1\ \{p \wedge t(i,j) < t_0\}$

fi
$\{p \wedge t(i,j) < t_0\}$
od
$\{p \wedge (i < n \wedge j < m \rightarrow b_1[i] = b_2[j])\}$
$\{q \equiv InRange(i,j) \wedge NoMatch(i,j) \wedge (i < n \wedge j < m \rightarrow b_1[i] = b_2[j])\}$

2. Create a program for binary search $BS(b,x)$ that searches for x (where $b[0] \leq x \leq b[n-1]$) in sorted array b whose $size(b) = n > 0$. In the postcondition, let L be the index of x if and only if x is in b, $0 \leq L < n$.

   - Let L and R be the left (inclusive) and right (exclusive) boundary of the searching range. We know that the idea of binary search is to "half" the search area after iteration (by letting middle point m be the new L or R in the next iteration) until the target x is found or the range is shrunk to size 0. We need to be careful while guaranteeing that the searching range shrinks after each iteration: because till the end, we can have $R = L+1$, then $m = \frac{L+R}{2} = L$ which can lead to divergence. Here let's look one way to solve this problem (we terminate the loop when the searching range has size 1):

     We let $L \coloneqq m$ or $R \coloneqq m$ in each iteration, then we need to terminate with $R = L+1$ (which means there is only $b[L]$ in the searching range) when x in not found. Because R can be $L+1$ and L can be $n-1$, we need to artificially define $b[n] = b[n-1]+1$.

     (As an aside, another design can be let $L \coloneqq m+1$ and $R \coloneqq m$ in each iteration. Think about how to create a loop using this design.)

   - The precondition of the program should be $p_0 \equiv Sorted(b) \wedge size(b) = n > 0 \wedge b[0] \leq x \leq b[n-1]$. Since $Sorted(b)$ is always true during the program and our searching procedure doesn't change it; we will only show it in the precondition and omit it everywhere else.

   - In the postcondition, to show whether we find x, we introduce a Boolean variable $found$. We say $found$ if and only if we have $b[L] = x$ at the end. The postcondition of the program can be written as $q \equiv 0 \leq L < n \wedge (b[L] = x \vee b[L] < x < b[L+1]) \wedge (found \leftrightarrow b[L] = x)$.

   - For loop invariant:
     - Simply dropping off either conjunct will cause a problem since the dropped off conjunct will be the negation of loop condition.
     - On the other hand, replacing $L+1$ by a variable R looks like a good idea, and we have $R \neq L+1$ while looping. The range of R should be $L+1 \leq R$ n, and we can end the loop with or
       .

- ❖ In the end, we can try to use loop invariant $p \equiv 0 \le L < R \le n \wedge (b[L] = x \vee b[L] < x < b[R]) \wedge (found \leftrightarrow b[L] = x)$
- ❖ At the same time, we find that we should try to use loop condition $B \equiv \neg found \wedge R \neq L + 1$.

- For the bound expression: since we want to increase L or decrease R while not $found$ and loop invariant says $R > L$, so we can try $R - L$.

- Then we can come up with the following partial program:

$\{p_0 \equiv Sorted(b) \wedge size(b) = n > 0 \wedge b[0] \le x \le b[n-1] < b[n]\}$
...
$\{\textbf{inv}\ p \equiv 0 \le L < R \le n \wedge (b[L] = x \vee b[L] < x < b[R]) \wedge (found \rightarrow b[L] = x)\}\ \{\textbf{bd}\ R - L\}$
while $\neg found \wedge R \neq L + 1$ **do**
    $\{p \wedge \neg found \wedge R \neq L + 1 \wedge R - L = t_0\}$
    $m \coloneqq (L + R) \div 2;$ # we can use ÷ for division, it has the same meaning as /
    $\{p_1 \equiv p \wedge \neg found \wedge R \neq L + 1 \wedge R - L = t_0 \wedge m = (L + R) \div 2\}$ # forward assignment
    if $b[m] = x$ **then**
        $\{p_1 \wedge b[m] = x\}\ found \coloneqq T; L \coloneqq m\ \{p \wedge R - L < t_0\}$
    else
        $\{p_1 \wedge b[m] \neq x\}$ ... $\{p \wedge R - L < t_0\}$
    fi
    $\{p \wedge R - L < t_0\}$

od
$\{0 \le L < R \le n \wedge (b[L] = x \vee b[L] < x < b[R]) \wedge (found \leftrightarrow b[L] = x) \wedge (found \vee R = L + 1)\}$
...
$\{q \equiv 0 \le L < n \wedge (b[L] = x \vee b[L] < x < b[L+1]) \wedge (found \leftrightarrow b[L] = x)\}$

- There are still gaps in the program and we don't have a full proof outline yet.
  - ❖ To get **inv** p from p_0, we need assign the starting values of L, R and $found$.
  - ❖ The true branch lacks proof: so, we need to add forward or backward assignments between statements. Here we are fine with a partial proof outline since the proof of assignments are syntactic.
  - ❖ $\{p \wedge (found \wedge R = L + 1)\}$ can imply q, so we don't need to add more statements.

  - ❖ For the false branch: we can use another conditional statement to add $L \coloneqq m$ or $R \coloneqq m$. Then we have the following partial proof outline under total correctness:

$\{p_0 \equiv Sorted(b) \wedge 1 \le n = size(b) \wedge b[0] \le x \le b[n-1] < b[n]\}$
$L \coloneqq 0; R \coloneqq n; found \coloneqq F;$
$\{\textbf{inv}\ p \equiv 0 \le L < R \le n \wedge (b[L] = x \vee b[L] < x < b[R]) \wedge (found \leftrightarrow b[L] = x)\}\ \{\textbf{bd}\ R - L\}$
while $\neg found \wedge R \neq L + 1$ **do**
    $\{p \wedge \neg found \wedge R \neq L + 1 \wedge R - L = t_0\}$
    $m \coloneqq (L + R) \div 2;$
    $\{p_1 \equiv p \wedge \neg found \wedge R \neq L + 1 \wedge R - L = t_0 \wedge m - (L + R) \div 2\}$
    if $b[m] = x$ **then**
        $\{p_1 \wedge b[m] = x\}\ found \coloneqq T; L \coloneqq m; \{p \wedge R - L < t_0\}$
    else
        $\{p_1 \wedge b[m] \neq x\}$ **if** $b[m] > x$ **then** $R \coloneqq m$ **else** $L \coloneqq m$ **fi** $\{p \wedge R - L < t_0\}$
    fi
    $\{p \wedge R - L < t_0\}$

od
$\{p \wedge (found \vee R = L + 1)\}$
$\{q \equiv 0 \leq L < n \wedge (b[L] = x \vee b[L] < x < b[L+1]) \wedge (found \leftrightarrow b[L] = x)\}$

3. Find the loop invariant for a loop that finds some x such that $x \leq sqrt(n) < x + 1$, where $n \geq 0$; or equivalently, $x^2 \leq n < (x+1)^2$. Let this expression be the post condition of the loop.

   1) How about replacing the "2" in the power with a variable k?
   If $p \equiv x^k \leq n < (x+1)^2$, then we end up with $k = 2$. What should be the other boundary of the range of the value of k? If the other boundary is larger than 2, then we most likely won't have $x^k < (x+1)^2$; if it is smaller than 2, then it is either 0 or 1, then this loop is trivial since there are only three possible values of k and I don't see it is helpful for looking for x. So, this is not a good idea.

   Replacing $(x+1)^2$ with $(x+1)^k$ also doesn't help us much about looking for x and we omit the discussion here. To sum up, *not every constant when replaced yields an invariant that works well*.

   2) It looks like that replacing the constant 1 with k is a good idea for loop invariant. We will end up with $k = 1$, and $k = n + 1$ is large enough as an upper bound for k, so k can have range $1 \leq k \leq n + 1$.

   In each iteration, to find some x, we need to get either x larger or k smaller, this implies that is a good bound expression.

   {inv $x^2 \leq n < (x+k)^2 \wedge 1 \leq k \leq n+1$**}{bd** $-x+k+n$**}**
   while $k \neq 1$ **do**
   … increase x or decrease k, and maybe something else …
   od
   $\{x^2 \leq n < (x+k)^2 \wedge 1 \leq k \leq n+1 \wedge k = 1\}$ # $p \wedge \neg B$
   $\{x^2 \leq n < (x+1)^2\}$

   ❖ We can try to use the idea of binary search to shrink the range: In each iteration we can compare n with the middle point $(x + k \div 2)^2$ to decide whether we want to increase the lower bound or the upper bound of the searching range. In addition, to add a precondition, we can start the search with $x = 0$ and $k = n + 1$. We can get the following partial proof outline.

   $\{0 \leq n < (n+1)^2\}\ x := 0; k := n+1;$
   {inv $p \equiv x^2 \leq n < (x+k)^2 \wedge 1 \leq k \leq n+1$**}{bd** $-x+k+n$**}**
   while $k \neq 1$ **do**
   $\{p \wedge k \neq 1 \wedge -x+k+n = t_0\}$
   if $(x + k \div 2)^2 > n$ **then**
   $k := k \div 2$
   else $\#(x + k \div 2)^2 \leq n$
   $x := x + k \div 2; k := k - k \div 2$
   $\{p \wedge -x+k+n < t_0\}$
   od
   $\{x^2 \leq n < (x+k)^2 \wedge 1 \leq k \leq n+1 \wedge k = 1\}$
   $\{x^2 \leq n < (x+1)^2\}$

   ❖ As an aside, we can see that k is decreasing in every iteration, we can simply use k instead of $-x + k + n$ as the bound expression.