Solution to Homework 7

CS 536: Science of Programming, Fall 2019

Lecture 18: Loop Convergence

1. (Bound expression for {**inv** p} {**bd** t} **while** j ≤ n **do** … j := j+1 **od**)

   a. `n-j` is invalid : It can be negative; `n-j` is decreased by the loop body.

   b. `n+j+C` is invalid: Incrementing `j` doesn't decrease it; `n+j+C` is nonnegative.

   c. `n-j+2*C` is valid. Incrementing `j` decreases it, and since C > 0, we know that n−j+ 2*C > n−j+C > 0.

2. (Complete full outline for total correctness) The answers are shown in red. (Parenthesized notes are just comments — you weren't expected to include them.) You didn't have to define p_0; if you didn't, you just have to write out the predicate. It also would have been fine to define other names to cut down on the writing; your choice.

```
{b[j] ≥ 1 ∧ 0 ≤ j < size(b)}                  // call this p₀
x := 1;
{p₀ ∧ x = 1}                                   // (sp of line above)
k := 0;
{p₀ ∧ x = 1 ∧ k = 0}                           // (sp of line above)
{inv p ≡ x = 2^k ≤ b[j] ∧ 0 ≤ j < size(b)}
{bd b[j]-x}                                    // (other bounds include b[j]-k, ceil(log₂(b[j]))-k)
while 2*x ≤ b[j] do
    {p ∧ 2*x ≤ b[j] ∧ b[j]-x = t₀}             // (let t₀ be value of bound expr at top of loop body)
    k := k+1;
    {p[k₀/k] ∧ 2*x ≤ b[j] ∧ b[j]-x = t₀ ∧ k = k₀+1}                    // (sp of line above)
    x := 2*x
    {p[k₀/k][x₀/x] ∧ 2*x₀ ≤ b[j] ∧ b[j]-x₀ = t₀ ∧ k = k₀+1 ∧ x = 2*x₀}   // (sp of line above)
    {p ∧ b[j]-x < t₀}      // (bound expr is now less than it was at top of loop body)
od
{p ∧ 2*x > b[j]}
{x = 2^k ≤ b[j] < 2^(k+1)}
```

Lecture 19: Finding Invariants [14 points]

3. (Candidate invariants for $q \equiv$ x ≥ 0 ∧ z = 2^x ≤ n < 2^(x+1) obtained by replacing a constant by a variable)

   a. (Replace the 0 in x ≥ 0 with a variable v.) For a range on v, making v ≥ 0 ensures we use nonnegative powers of 2. However, there's no good way to initialize x; it's as hard as the problem we're trying to solve. E.g., initializing x to 0 means z = 1 so we need to assume n ≥ 1. But if z = 1 then we need n < 2², which we don't know.

$\{n \geq 1\}$ *initialization ???* $\{$**inv** $x \geq v \wedge z = 2\hat{\ }x \leq n < 2\hat{\ }(x+1) \wedge v \geq 0\}$ **while** $v \neq 0 \ldots$

b. (Replace the n in $2\hat{\ }x \leq n < 2\hat{\ }(x+1)$ with a variable v). This might work: the loop body can increment v; if it now equals $2\hat{\ }(x+1)$ then increment x to get v back in range.

$\{n \geq 1\}$ x := 0; z := 1; v := 1; $\{$**inv** $x \geq 0 \wedge z = 2\hat{\ }x \leq v < 2\hat{\ }(x+1) \wedge v \geq 1\}$ **while** $v \neq n \ldots$

c. (Replace the 1 in $2\hat{\ }(x+1)$ with a variable v). The range $v \geq 1$ seems plausible; initializing x to 0 and z to 1 but we need something large to initialize v to. One initialization is v := n, which works, since $2\hat{\ }n > n$ for all $n \geq 1$. But how do we calculate $2\hat{\ }n$ in order to know that $n < 2\hat{\ }n$? If we can actually use the (^) operation, then it's easy. If not, we'd need some sort of a loop to calculate $2\hat{\ }0$, $2\hat{\ }1, \ldots, 2\hat{\ }n$, which is as hard as the overall problem.

$\{n \geq 1\}$ x := 0; z := 1; v := 1; *some variable := 2^n somehow???*

$\{$**inv** $x \geq 0 \wedge z = 2\hat{\ }x \leq n < 2\hat{\ }(x+v)\}$ **while** $v \neq 1 \ldots$

4. (Same $q \equiv x \geq 0 \wedge z = 2\hat{\ }x \leq n < 2\hat{\ }(x+1)$, but look for variants obtained by dropping a conjunct.)

a. (Drop $x \geq 0$) Doesn't work: How can we have $x < 0$ and $z = 2\hat{\ }x \leq n < 2\hat{\ }(x+1)$ together? Also, what do we initialize x to?

$\{n \geq 1\}$??? $\{$**inv** $z = 2\hat{\ }x \leq n < 2\hat{\ }(x+1)\}$ **while** $x < 0 \ldots$

b. (Drop $2\hat{\ }x \leq n$) How do we initialize x and z? x needs to be large enough to guarantee $2\hat{\ }x \geq n$, otherwise the loop test will fail. But as in 3(c), calculating $2\hat{\ }x$ for large x is as hard as the original problem.

$\{n \geq 1\}$??? $\{$**inv** $x \geq 0 \wedge z = 2\hat{\ }x \wedge n < 2\hat{\ }(x+1)\}$ **while** $2\hat{\ }x > n$

c. (Drop $n < 2\hat{\ }(x+1)$) (Finally, something that works.) We can initialize x := 0; z := 1 and use $n < 2*z$ as equivalent to our loop test.

$\{n \geq 1\}$ x := 0; z := 1 $\{$**inv** $x \geq 0 \wedge z = 2\hat{\ }x < n < 2\hat{\ }(x+1)\}$ **while** $n < 2*z$

Lecture 20: Array Assignments [14 points]

5. (*wp* of array assignments)

a. $\{p\}$ b[j] := a; $\{q\}$ b[i] := c $\{b[j] \leq b[i]\}$

$q \equiv wp(b[i] := c, b[j] \leq b[i]) \equiv (b[j] \leq b[i])[c/b[i]]$

$\equiv b[j][c/b[i]] \leq b[i][c/b[i]] \equiv$ **if** $j = i$ **then** c **else** b[j] **fi** $\leq c$

$\Leftrightarrow$ **if** $j = i$ **then** $c \leq c$ **else** $b[j] \leq c$ **fi** // Note: $\Leftrightarrow$ here, not $\equiv$

$\Leftrightarrow j = i \vee b[j] \leq c$ // or anything equivalent, like $i = j \vee b[j] \leq c$ or $i \neq j \rightarrow b[j] \leq c$

$p \equiv wp(b[j] := a, i = j \vee b[j] \leq c)$

$\equiv (i = j \vee b[j] \leq c)[a/b[j]]$

$\equiv i = j \vee (b[j]\,[a/b[j]]) \leq c$

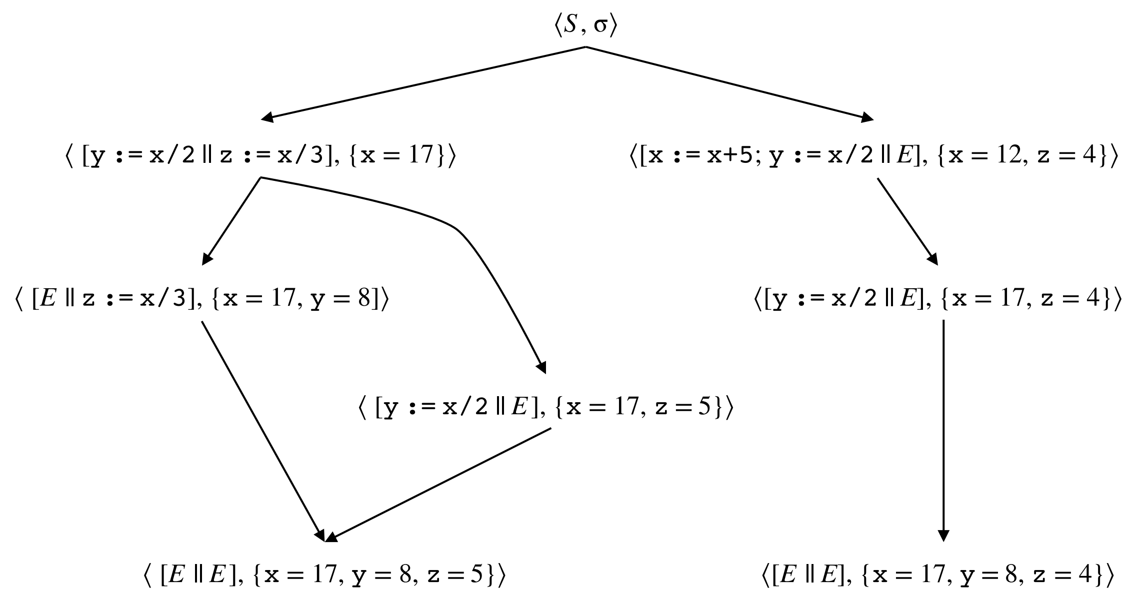
$\equiv i = j \vee a \leq c$ // or anything equivalent, like $i \neq j \rightarrow a \leq c$

b. $\{p\}$ b[j] := b[m]; $\{q\}$ b[k] := b[n] $\{b[j] < b[k] \wedge j \neq k\}$

$$
\begin{aligned}
q &\equiv wp(\mathtt{b[k]} := \mathtt{b[n]},\ \mathtt{b[j]} < \mathtt{b[k]} \wedge \mathtt{j} \neq \mathtt{k}) \\
&\equiv (\mathtt{b[j]} < \mathtt{b[k]} \wedge \mathtt{j} \neq \mathtt{k})[\mathtt{b[n]}/\mathtt{b[k]}] \\
&\equiv \mathtt{b[j]}[\mathtt{b[n]}/\mathtt{b[k]}] < \mathtt{b[k]}[\mathtt{b[n]}/\mathtt{b[k]}] \wedge \mathtt{j} \neq \mathtt{k} \\
&\equiv \textbf{if } \mathtt{j} = \mathtt{k} \textbf{ then } \mathtt{b[n]} \textbf{ else } \mathtt{b[j]} \textbf{ fi} < \textbf{if } \mathtt{k} = \mathtt{k} \textbf{ then } \mathtt{b[n]} \textbf{ else } \mathtt{b[k]} \textbf{ fi} \wedge \mathtt{j} \neq \mathtt{k} \\
&\equiv \textbf{if } \mathtt{j} = \mathtt{k} \textbf{ then } \mathtt{b[n]} \textbf{ else } \mathtt{b[j]} \textbf{ fi} < \mathtt{b[n]} \wedge \mathtt{j} \neq \mathtt{k} && \text{// (since } \mathtt{k} = \mathtt{k}\text{)} \\
&\equiv \mathtt{b[j]} < \mathtt{b[n]} \wedge \mathtt{j} \neq \mathtt{k} && \text{// (since } \mathtt{j} \neq \mathtt{k}\text{)} \\
p &\equiv wp(\mathtt{b[j]} := \mathtt{b[m]},\ \mathtt{b[j]} < \mathtt{b[n]} \wedge \mathtt{j} \neq \mathtt{k}) \\
&\equiv \mathtt{b[j]}[\mathtt{b[m]}/\mathtt{b[j]}] < \mathtt{b[n]}[\mathtt{b[m]}/\mathtt{b[j]}] \wedge \mathtt{j} \neq \mathtt{k} \\
&\Leftrightarrow \mathtt{b[m]} < \mathtt{b[n]}[\mathtt{b[m]}/\mathtt{b[j]}] \wedge \mathtt{j} \neq \mathtt{k} && \text{// (optimizing } \mathtt{b[j]}[\mathtt{b[m]}/\mathtt{b[j]}]\text{)} \\
&\Leftrightarrow \mathtt{b[m]} < \textbf{if } \mathtt{n} = \mathtt{j} \textbf{ then } \mathtt{b[m]} \textbf{ else } \mathtt{b[n]} \textbf{ fi} \wedge \mathtt{j} \neq \mathtt{k} \\
&\Leftrightarrow \textbf{if } \mathtt{n} = \mathtt{j} \textbf{ then } \mathtt{b[m]} < \mathtt{b[m]} \textbf{ else } \mathtt{b[m]} < \mathtt{b[n]} \textbf{ fi} \wedge \mathtt{j} \neq \mathtt{k} \\
&\Leftrightarrow \textbf{if } \mathtt{n} = \mathtt{j} \textbf{ then } \mathtt{F} \textbf{ else } \mathtt{b[m]} < \mathtt{b[n]} \textbf{ fi} \wedge \mathtt{j} \neq \mathtt{k} \\
&\Leftrightarrow \mathtt{n} \neq \mathtt{j} \wedge \mathtt{b[m]} < \mathtt{b[n]} \vee \mathtt{j} \neq \mathtt{k}
\end{aligned}
$$

Lecture 21: Parallel Programs [16 points]

6. We have $S \equiv [\mathtt{x := x+5;\ y := x/2} \parallel \mathtt{z := x/3}]$ and $\sigma = \{\mathtt{x} = 12\}$. From the evaluation graph below, $M(S, \sigma) = \{\{\mathtt{x} = 17, \mathtt{y} = 8, \mathtt{z} = 5\}, \{\mathtt{x} = 17, \mathtt{y} = 8, \mathtt{z} = 4\}\}$



7. (Pairwise disjoint programs and condition check) Our threads are

- {x ≠ y} `x := u ; y := u` {x = y}
- {v = z} `z := z+1 ; v := v+1` {v = z}, and
- {w ≥ u+x} `w := u+1; w := v` {w > u+x}

i	*j*	*Change i*	*Vars j*	*Free j*	*Disjoint Program?*	*Disjoint Conditions?*
1	2	x y	v z	v z	Yes	Yes
1	3	x y	u v w	u x w	Yes	No
2	1	v z	u x y	x y	Yes	Yes
2	3	v z	u v w	u x w	No	Yes
3	1	w	u x y	x y	Yes	Yes
3	2	w	v z	v z	Yes	Yes

