CSE Template

Rohan Mukherjee

July 25, 2024

1. (a) We use forward reasoning in the following way:

{{$c > 0$ `and` $s > 0$}}

```
    e = 5n * c + 1;
```

{{$c > 0$ `and` $s > 0$ `and` $e = 5c + 1$}}

```
s = s + 3n;
```

{{$c > 0$ `and` $s - 3 > 0$ `and` $e = 5c + 1$}}

```
    c = c * s;
```

{{$c/s > 0$ `and` $s > 3$ `and` $e = 5c/s + 1$}}

{{e > 1}}

We prove that last implication. We have that $c/s > 0$, so we know since c/s is an integer that $c/s \geq 1$. Thus $e = 5c/s + 1 \geq 5 \cdot 1 + 1 = 6 > 1$. Thus $e > 1$.

(b) {{$s \geq 0$ `and` $e = 0$}}

{{$3s + 2 \geq e + 2$}}

```
s = 3n * s;
```

{{$s + 2 \geq e + 2$}}

```
e = e+2n;
```

{{$s + 2 \geq e$}}

```
c = s+2n;
```

{{$c \geq e$}}

The implication can be proved as follows. Since $s \geq 0$, we know that $3s+2 \geq 3 \cdot 0+2 = 2 \geq 0+2 = e = 2$, since $e = 0$. Thus we have that $3s + 2 \geq e + 2$ from the precondition and we are done.

(c) {{$c \geq 1$ `and` $s = c^2$}}

```
if (s < 20n) {
```

{{$c \geq 1$ `and` $s = c^2$ `and` $s < 20$}}

```
    s = s + 5n;
```

{{$c \geq 1$ `and` $s - 5 = c^2$ `and` $s - 5 < 20$}}

```
} else if (s < 30n) {
```

```
    {{c ≥ 1 and s = c^2 and 20 ≤ s < 30}}
    s = (s/c)+1n;
    {{c ≥ 1 and c(s − 1) = c^2 and 20 ≤ c(s − 1) < 30}}
} else {
    {{c ≥ 1 and s = c^2 and s ≥ 30}}
    s = s/c;
    {{c ≥ 1 and cs = c^2 and cs ≥ 30}}
}
{{c ≥ 1 and s − 5 = c^2 and s − 5 < 20
or c ≥ 1 and c(s − 1) = c^2 and c(s − 1) < 30
or c ≥ 1 and cs = c^2 and cs ≥ 30}}
{{s > 5}}
```

We prove the final implication by cases. In the first case, we know that $c \geq 1$, $s - 5 = c^2$ and $s - 5 < 20$. Using the second condition, we see that $s - 5 = c^2$ and so $s = 5 + c^2$. Since $c \geq 1$, we know that $c^2 \geq 1$. Using this we see that $s = 5 + c^2 \geq 5 + 1 = 6 > 5$.

In the second case, we have that $c \geq 1$ and $c(s - 1) = c^2$ and $20 \leq c(s - 1) < 30$. The second condition tells us that $s - 1 = c$ after dividing by c (c is nonzero since by the first condition $c \geq 1$). We know from the last condition that $c(s - 1) \geq 20$. Using that $s - 1 = c$ now tells us that $c^2 \geq 20$. Taking square roots on both sides, noting that $c \geq 1$ and hence $c \geq 0$, we have that $c \geq \sqrt{20} > 4$. Since c is an integer, we must have that $c \geq 5$. Since $s = c + 1$, we have that $s \geq 5 + 1 = 6 > 5$, and we are done.

In the last case, we have that $c \geq 1$ and $cs = c^2$ and $cs \geq 30$. Once again using the second condition tells us that $s = c$ because $c \geq 1$ so we can divide on both sides by the nonzero constant c. Plugging this into $cs \geq 30$ tells us that $s^2 \geq 30$. Since $s = c$ we have that $s \geq 1$ as well. Taking square roots on both sides of $s^2 \geq 30$ tells us that $s \geq \sqrt{30} > 5$. Thus $s > 5$ as well.

2. (a) Initially, we know that $x = x_0$, so we have that $4y = 4 \cdot 0 = 0 = x_0 - x$. Similarly, since $x_0 \geq 0$ we know that $x = x_0 \geq 0 \geq -4$, so the invariant holds at the top of the loop.

(b) We use forward reasoning. We have:

```
{{Inv: 4y = x₀ − x and x ≥ −4}}
while (x >= 0) {
    {{4y = x₀ − x and x ≥ −4 and x ≥ 0}}  ⟺  {{4y = x₀ − x and x ≥ 0}}
    y = y + 1n;
    {{4(y − 1) = x₀ − x and x ≥ 0}}
    x = x - 4n;
    {{4(y − 1) = x₀ − (x + 4) and x + 4 ≥ 0}}
}
```

The fact that the last condition proves the invariant can be shown as follows. $4(y-1) = x_0 - (x+4)$ is the same as $4y - 4 = x_0 - x - 4$, and adding 4 to both sides shows that $4y = x_0 - x$ as required. The second condition is $x + 4 \geq 0$ which is the same as $x \geq -4$.

(c) Once we have exited the loop we will have that $x < 0$. We can use the invariant to see then that $-4 \leq x < 0$. Using the fact that $4y = x_0 - x$, we can use the fact that $x \geq -4$ to then multiply both sides by -1 to get that $-x \leq 4$. Thus $4y = x_0 - x \leq x_0 + 4$. Since $x < 0$ here, and since $4y = x_0 - x$, we have again that $-x > 0$ after multiplying both sides by -1, so we can apply this to see that $4y = x_0 - x > x_0$. Thus $x_0 < 4y$ and we are done.

3. (a) Initially, we have that $a = 0$, $b = 0$, and $L = L_0$. So,

$$\begin{aligned} \mathrm{sum-gt(L_0, x)} &= \mathrm{sum-gt(L, x)} && L = L_0 \\ &= \mathrm{a + sum-gt(L, x)} && a = 0 \end{aligned}$$

Similarly, we have that:

$$\begin{aligned} \mathrm{sum-lt(L_0, x)} &= \mathrm{sum-lt(L, x)} && L = L_0 \\ &= \mathrm{b + sum-lt(L, x)} && b = 0 \end{aligned}$$

So the invariant holds at the top of the loop.

(b)

```
{{Inv: sum-gt(L₀,x) = a + sum-gt(L, x) and
sum-lt(L₀,x) = b + sum-lt(L, x)}}
while (L !== nil) {
    {{sum-gt(L₀,x) = a + sum-gt(L, x) and
    sum-lt(L₀,x) = b + sum-lt(L, x) and
    L = cons(L.hd, L.tl)}}
    if (L.hd > x) {
        {{sum-gt(L₀,x) = a + sum-gt(L, x) and
```

```
        sum-lt(L_0,x) = b + sum-lt(L, x) and
        L = cons(L.hd, L.tl) and L.hd > x}}
        a = a + (L.hd - x);
        {{sum-gt(L_0,x) = a - (L.hd - x) + sum-gt(L, x) and
        sum-lt(L_0,x) = b + sum-lt(L, x) and
        L = cons(L.hd, L.tl) and L.hd > x}}
    } else if (L.hd < x) {
        {{sum-gt(L_0,x) = a + sum-gt(L, x) and
        sum-lt(L_0,x) = b + sum-lt(L, x) and
        L = cons(L.hd, L.tl) and L.hd < x}}
        b = b + (x-L.hd);
        {{sum-gt(L_0,x) = a + sum-gt(L, x) and
        sum-lt(L_0,x) = b - (x-L.hd) + sum-lt(L, x) and
        L = cons(L.hd, L.tl) and L.hd < x}}
    } else {
        // Do nothing
        {{sum-gt(L_0,x) = a + sum-gt(L, x) and
        sum-lt(L_0,x) = b + sum-lt(L, x) and
        L = cons(L.hd, L.tl) and L.hd = x}}
    }
    {{sum-gt(L_0,x) = a - (L.hd - x) + sum-gt(L, x) and
        sum-lt(L_0,x) = b + sum-lt(L, x) and
        L = cons(L.hd, L.tl) and L.hd > x or
        sum-gt(L_0,x) = a + sum-gt(L, x) and
        sum-lt(L_0,x) = b - (x-L.hd) + sum-lt(L, x) and
        L = cons(L.hd, L.tl) and L.hd < x or
        sum-gt(L_0,x) = a + sum-gt(L, x) and
        sum-lt(L_0,x) = b + sum-lt(L, x) and
        L = cons(L.hd, L.tl) and L.hd = x}}
    ⟹
    {{sum-gt(L_0,x) = a + sum-gt(L.tl, x) and
    sum-lt(L_0,x) = b + sum-lt(L.tl, x)}}
    L = L.tl;
    {{sum-gt(L_0,x) = a + sum-gt(L, x) and
    sum-lt(L_0,x) = b + sum-lt(L, x)}}
}
```

We used backwards reasoning for the last line L = L.tl at the end there. I put the invariant under that line to make it clear how I am applying the backwards reasoning, and I added an $\implies$ to show the implication that we have to prove. We prove that now by cases.

In the first case, we have that sum-gt(L_0, x) = a - (L.hd -x) + sum-gt(L, x) and sum-lt(L_0, x) = b + sum-lt(L, x) and L = cons(L.hd, L.tl) and L.hd > x. We see that:

$$\begin{aligned} \mathrm{sum-gt(L,x)} &= \mathrm{sum-gt(cons(L.hd, L.tl), x)} && \text{L = cons(L.hd, L.tl)} \\ &= \mathrm{(L.hd - x) + sum - gt(L.tl, x)} && \text{L.hd} > x \text{ and def of sum-gt} \end{aligned}$$

Plugging this calculation in, we see that sum-gt(L_0, x) = a - (L.hd - x) + sum-gt(L, x) = a - (L.hd - x) + (L.hd - x) + sum-gt(L.tl, x) = a + sum-gt(L.tl, x) as desired. Similarly,

$$\begin{aligned} \mathrm{sum-lt(L,x)} &= \mathrm{sum-lt(cons(L.hd, L.tl), x)} && \text{L = cons(L.hd, L.tl)} \\ &= \mathrm{sum-lt(L.tl, x)} && \text{L.hd} > x \text{ and def of sum-lt} \end{aligned}$$

So we can plug this into the above to see that sum-lt(L_0, x) = b + sum-lt(L, x) = b + sum-lt(L.tl, x) as well. This completes the proof of this case.

In the second case, we know that sum-gt(L_0, x) = a + sum-gt(L, x) and sum-lt(L_0, x) = b - (x-L.hd) + sum-lt(L, x) and L = cons(L.hd, L.tl) and L.hd < x. We see that:

$$\begin{aligned} \mathrm{sum-gt(L,x)} &= \mathrm{sum-gt(cons(L.hd, L.tl), x)} && \text{L = cons(L.hd, L.tl)} \\ &= \mathrm{sum-gt(L.tl, x)} && \text{L.hd} < x \text{ and def of sum-gt} \end{aligned}$$

Plugging this calculation in, we see that sum-gt(L_0, x) = a + sum-gt(L, x) = a + sum-gt(L.tl, x) as desired. Similarly,

$$\begin{aligned} \mathrm{sum-lt(L,x)} &= \mathrm{sum-lt(cons(L.hd, L.tl), x)} && \text{L = cons(L.hd, L.tl)} \\ &= \mathrm{(x - L.hd) + sum - lt(L.tl, x)} && \text{L.hd} < x \text{ and def of sum-lt} \end{aligned}$$

So we can plug this into the above to see that sum-lt(L_0, x) = b - (x-L.hd) + sum-lt(L, x) = b + sum-lt(L.tl, x) as well. This completes the proof of this case.

In the last case, we know that sum-gt(L_0, x) = a + sum-gt(L, x) and sum-lt(L_0, x) = b + sum-lt(L, x) and L = cons(L.hd, L.tl) and L.hd = x. We see that:

$$\begin{aligned} \mathrm{sum-gt(L,x)} &= \mathrm{sum-gt(cons(L.hd, L.tl), x)} && \text{L = cons(L.hd, L.tl)} \\ &= \mathrm{sum-gt(L.tl, x)} && \text{L.hd} = x \text{ and def of sum-gt} \end{aligned}$$

Plugging this calculation in, we see that sum-gt(L_0, x) = a + sum-gt(L, x) = a + sum-gt(L.tl, x) as desired. Similarly,

$$\begin{aligned} \mathrm{sum-lt(L,x)} &= \mathrm{sum-lt(cons(L.hd, L.tl), x)} && \text{L = cons(L.hd, L.tl)} \\ &= \mathrm{sum-lt(L.tl, x)} && \text{L.hd} = x \text{ and def of sum-lt} \end{aligned}$$

So we can plug this into the above to see that sum-lt(L_0, x) = b + sum-lt(L, x) = b + sum-lt(L.tl, x) as well. This completes the proof of this case and hence the loop invariant.

(c) When we exit the loop, we know that L = nil, that sum-gt(L_0, x) = a + sum-gt(L, x) and sum-lt(L_0, x) = b + sum-lt(L,x). Recalling that sum-gt(nil, x) = 0 and sum-lt(nil, x) = 0, we see that sum-gt(L_0, x) = a + 0 = a and sum-lt(L_0, x) = b + 0 = b. Thus we have that sum-gt(L_0, x) = a and sum-lt(L_0, x) = b as required which completes the proof.

5. (a) Let $P(L)$ for a list L be defined as $\text{contains}(a, \text{concat}(L, S)) = \text{contains}(a, L) \vee \text{contains}(a, S)$. We prove this claim by structural induction on L. First, if L = nil, then we have that:

$$\begin{aligned} \text{contains}(a, \text{concat}(L, S)) &= \text{contains}(a, \text{concat}(\text{nil}, S)) && \text{L=nil} \\ &= \text{contains}(a, S) && \text{def of concat} \end{aligned}$$

Since by definition contains(a, L) = contains(a, nil) = False, we know that contains(a, L) or contains(a, S) = contains(a, S) which shows the base case holds. Now suppose it holds for some list L. Then we have that:

$$\begin{aligned} \text{contains}(a, \text{concat}(\text{cons}(x, L), S)) &= \text{contains}(a, \text{cons}(x, \text{concat}(L, S))) && \text{def of concat} \\ &= (a = x) \vee \text{contains}(a, \text{concat}(L, S)) && \text{def of contains} \\ &= ((a = x) \vee \text{contains}(a, L)) \vee \text{contains}(a, S) && \text{I.H.} \\ &= \text{contains}(a, \text{cons}(x, L)) \vee \text{contains}(a, S) && \text{def of contains} \end{aligned}$$

This completes the proof by structural induction of this first claim.

(b) Let the claim $P(U)$ for a BST U be defined as contains(a, toList(U)) = (search(a, U) $\neq$ undefined). We prove this claim by structural induction on U. First, if U is empty, then we have that:

$$\begin{aligned} \text{contains}(a, \text{toList}(U)) &= \text{contains}(a, \text{toList}(\text{empty})) && \text{U=empty} \\ &= \text{contains}(a, \text{nil}) && \text{def of toList} \\ &= \text{false} && \text{def of contains} \end{aligned}$$

Similarly,

$$\begin{aligned} \text{search}(a, U) &= \text{search}(a, \text{empty}) && \text{U=empty} \\ &= \text{undefined} && \text{def of search} \end{aligned}$$

So $\text{search}(a, U) \neq \text{undefined}$ is false as well. Thus the base case holds. We now prove that

```
contains(a, toList(node(b, S, T))) = contains(a, toLisT(S)) or (a == b)
                                      or contains(a, toList(T))
```

Recall that toList(node(b, S, T)) = concat(toList(S), cons(b, toList(T))). Then by part (a), we have that:

$$\begin{aligned} \text{contains}(a, \text{toList}(\text{node}(b, S, T))) &= \text{contains}(a, \text{concat}(\text{toList}(S), \text{cons}(b, \text{toList}(T)))) && \text{def toList} \\ &= \text{contains}(a, \text{toList}(S)) \vee \text{contains}(a, \text{cons}(b, \text{toList}(T))) && \text{part (a)} \\ &= \text{contains}(a, \text{toList}(S)) \vee (a = b) \vee \text{contains}(a, \text{toList}(T)) && \text{def contains} \end{aligned}$$

Now suppose that $P(S), P(T)$ hold for BSTs S, T.

First suppose that $a < b$. By the BST invariant, nothing $> b$ is in S and nothing $\leq b$ is in T. In particular, T does not have a node with value a, since all of its nodes have value $> b$. Thus we see that contains(a, toList(T)) =false. Similarly, since $a < b$ we have that $a \neq b$. So the above in this case equals contains(a, toList(S)), which is by the IH search(a, S) $\neq$ undefined. By the definition of search, we have that search(a, node(b, S, T)) = search(a, S) since $a < b$. So we put these expressions together to get that the above equals search(a, node(b, S, T)) $\neq$ undefined as desired.

In the second case, where $a = b$, the above expression would evaluate to true. Similarly, search(a, node(b, S, T)) = node(a, S, T) by definition. This is by definition undefined, so
(search(a, node(b, S, T)) $\neq$ undefined) is true as well. This completes the proof in this case.

In the last case, suppose that $a > b$. Once again, by the BST invariant, nothing $> a$ appears as a node of S. So contains(a, toList(S)) =false. Similarly $a \neq b$, so in this case this simplifies to contains(a, toList(T)). By the IH, this is search(a, T) $\neq$ undefined. By the definition of search, search(a, search(b, S, T)) = search(a, T) since $a > b$. So this expression also equals search(a, node(b, S, T)) $\neq$ undefined as required.