CSE 331: Software Design & Implementation Fall 2025

Homework 4

Due: Friday, October 24th, 6pm

Task 1 – Off To the Cases! [26 pts]

In this problem, you will practice proving correctness of straight-line code using **backward reasoning**.

a) Use **forward** reasoning to fill in the assertions. Then, show that the code is correct by proving by **cases** that P implies Q.

$\{\{\, x \geqslant 0 \,\}\}$
```
if (x >= 8) {
```
$\{\{$ _____ $\}\}$
```
  y = 3 * x - 9;
```
$\{\{$ _____ $\}\}$
```
} else {
```
$\{\{$ _____ $\}\}$
```
  y = 20 - x;
```
$\{\{$ _____ $\}\}$
```
}
```
$\{\{\, P:$ _____ or _____ $\}\}$
$\{\{\, Q:\ y \geqslant 12 \,\}\}$

b) Use **backward** reasoning to fill in the assertions.

Fill in each blank by applying the rules *exactly* as taught in lecture. Then, if you want, you can simplify the resulting assertion, but do not weaken it. (Use "↔" to separate assertions as before.)

Then, show that the code is correct by proving by cases that P_1 implies Q_1 and P_2 implies Q_2. Note: the older version asked to prove the implications above the if branch, which required a step of forward reasoning. This has been changed.

$\{\!\{ P: \ x \geqslant 0 \}\!\}$
```
if (x >= 8) {
```
$\{\!\{ P_1 : x >= 8 \}\!\}$
$\{\!\{ Q_1 : \_\_\_\_\_ \}\!\}$
```
  y = 3 * x - 9;
```
$\{\!\{ \_\_\_\_\_ \}\!\}$
```
} else {
```
$\{\!\{ P_2 : x < 8 \}\!\}$
$\{\!\{ Q_2 : \_\_\_\_\_ \}\!\}$
```
  y = 20 - x;
```
$\{\!\{ \_\_\_\_\_ \}\!\}$
```
}
```
$\{\!\{ y \geqslant 12 \}\!\}$

c) Are the calculations you did in (b) the same as the ones in (a)? Why are they the same or not the same?

d) Use **forward** reasoning to fill in assertions P_1 and P_2 and **backward** reasoning to fill in assertions Q_1 and Q_2. Then, show that the code is correct by proving by **calculation** that P_1 implies Q_1 and that P_2 implies Q_2.

$\{\!\{ x \geqslant 0 \}\!\}$
```
if (x >= 8) {
```
$\{\!\{ P_1 : \_\_\_\_\_ \}\!\}$
$\{\!\{ Q_1 : \_\_\_\_\_ \}\!\}$
```
  y = 3 * x - 9;
```
$\{\!\{ \_\_\_\_\_ \}\!\}$
```
} else {
```
$\{\!\{ P_2 : \_\_\_\_\_ \}\!\}$
$\{\!\{ Q_2 : \_\_\_\_\_ \}\!\}$
```
  y = 20 - x;
```
$\{\!\{ \_\_\_\_\_ \}\!\}$
```
}
```
$\{\!\{ y \geqslant 12 \}\!\}$

e) Proving the correctness in (d), where we mixed forward and backward reasoning, did not require cases. Why was it not necessary in this case? Did not having cases save the number of calculations?

Task 2 – Loops, I Did It Again [13 pts]

Suppose we define the set of binary digits as

$$\textbf{type}\ \text{Digit} := 0 \mid 1$$

Then, we can represent a number written in binary as the following list type:

$$\textbf{type}\ \text{Digits} := \text{last}(\text{Digit}) \mid \text{cons}(\text{Digit}, \text{Digits})$$

The following function, value : Digits $\to \mathbb{Z}$ translates a sequence of binary digits into the corresponding integer value:

$$\begin{aligned} \text{value}(\text{last}(d)) &:= d \\ \text{value}(\text{cons}(d, R)) &:= d + 2\,\text{value}(R) \end{aligned}$$

For example, we can see that

$$\begin{aligned} &\text{value}(\text{cons}(1, \text{cons}(1, \text{last}(0)))) \\ &= 1 + 2\,\text{value}(\text{cons}(1, \text{last}(0))) \\ &= 1 + 2(1 + 2\,\text{value}(\text{last}(0))) \\ &= 1 + 2(1 + 2 \cdot 0) \\ &= 1 + 2(1 + 0) \\ &= 1 + 2 \\ &= 3 \end{aligned}$$

Note that the binary representation of 3 is usually written 011, not 110, so this definition expects the digits to be stored in the list in reversed order, with the *least* significant digit at the front. Such a representation is called "little endian" (while the ordinary representation is "big endian").

We can represent the Digits data type as this Java class:

```
public class Digits {
    public int digit;
    public Digits next;
}
```

where `next` is `null` if the node is a "last" and non-`null` if it is a "cons".

The following code claims to be the value of the digits in the variable `L` of type `Digits`:

$\{\!\{ L = L_0 \}\!\}$
`int b = 1;`
`int v = 0;`
$\{\!\{$ Inv: value$(L_0) = v + b \cdot$ value$(L) \}\!\}$
`while (L.next != null) {`
  $\{\!\{ P_1 \colon\ L =$ cons(d, R) and $\ldots \}\!\}$
  `v = v + b * L.digit;`
  `b = 2 * b;`
  `L = L.next;`
`}`
$\{\!\{ P_2 \colon\ L =$ last(d) and $\ldots \}\!\}$
`v = v + b * L.digit;`
$\{\!\{ v =$ value$(L_0) \}\!\}$

Inside the body of the loop, the assertion will start with $L =$ cons(d, R) in order to encode the fact that `L.next != null`. The former is the mathematical equivalent of the latter code. Likewise, after the loop, the assertion will start with $L =$ last(d) as that is the mathematical equivalent of the fact that `L.next == null`. The variable d refers to the digit stored in `L.digit` and R refers to `L.next`.

a) Prove that the invariant is true when we get to the top of the loop the first time.

b) What are the known facts at P_2, when we exit the loop?

c) Prove that the postcondition holds. Use either forward or backward reasoning (your choice) on the line of code after the loop and then check that the resulting implication holds.

Remember that `L.digit` is called "d" in the math.

d) What are the known facts at P_1, when we enter the loop body?

e) Prove that the invariant is preserved by the body of the loop. Use either forward or backward reasoning (your choice) to reduce the body to an implication and then check that it holds.

Remember that `L.next` is called "R" in the math.

Task 3 – Loop Dreams [11 pts]

In this problem, we will prove the correctness one version of the code for `log2` produced by AI in Homework 1. Here is the code produced by Cursor's chat agent:

$\{\{\, n \geqslant 1 \,\}\}$
```
int k = 0;
int m = 1;
```
$\{\{$ Inv: $m = 2^k$ and $m < 2n \,\}\}$
```
while (m < n) {
    m = 2 * m;
    k = k + 1;
}
```
$\{\{\, 2^{k-1} < n$ and $n \leqslant 2^k \,\}\}$

The postcondition comes from the specification, specifically the `@return` tag, which said that the value of k returned should satisfy these two inequalities.

The loop invariant was not given in the specification, nor was it provided by the AI. I have filled it in for you to make it possible to complete the proof. (More on this later...)

a) Prove that the invariant is true when we get to the top of the loop the first time.

b) Prove that, when we exit the loop, the postcondition holds.

c) Prove that the invariant is preserved by the body of the loop. Use either forward or backward reasoning (your choice) to reduce the body to an implication and then check that it holds.

d) Would you have guessed that was the loop invariant just by looking at the code?

If the author of the code (AI, in this case) had proven the code correct, do you think they should have included the loop invariant in the comments or is it fine for them to leave it out and let you figure it out for yourself?

Task 4 – Extra Credit: Sum-thing Borrowed, Sum-thing Blue [16 pts]

The function mult : ($\mathbb{Z}$, List) $\to$ List multiplies each element in a list of integers by a given multiplier:

$$\begin{aligned} \mathsf{mult}(x, \mathsf{nil}) &:= \mathsf{nil} \\ \mathsf{mult}(x, m :: L) &:= x \cdot m :: \mathsf{mult}(x, L) \end{aligned}$$

Similar to the Digits data type in Task 2, we represent our IntList as this Java class:

```
public class IntList {
    public int hd;
    public IntList tl;
}
```

The following two claim to be the sum of an IntList whose elements were multiplied by a factor of 3:

A.

$\{\!\{ L = L_0 \}\!\}$
```
int z = 0;
int x = 3;
```
$\{\!\{$ Inv: $\mathsf{sum}(\mathsf{mult}(x, L_0)) = z + \mathsf{sum}(\mathsf{mult}(x, L)) \}\!\}$
```
while (L.tl != null) {
```
$\{\!\{ P_1 :$ _____ $\}\!\}$

$\{\!\{ Q_1 :$ _____ $\}\!\}$
```
    z = z + L.hd * x;
```
$\{\!\{ Q_0 :$ _____ $\}\!\}$
```
    L = L.tl;
}
```
$\{\!\{ z = \mathsf{sum}(\mathsf{mult}(x, L)) \}\!\}$

B.

$\{\!\{ L = L_0 \}\!\}$
```
int d = 0;
int v = 3;
```
$\{\!\{$ Inv: $\mathsf{sum}(L_0) = d + \mathsf{sum}(L) \}\!\}$
```
while (L.tl != null) {
```
$\{\!\{ P_1 :$ _____ $\}\!\}$

$\{\!\{ Q_1 :$ _____ $\}\!\}$
```
    d = d + L.hd;
```
$\{\!\{ Q_0 :$ _____ $\}\!\}$
```
    L = L.tl;
}
d = d * v
```
$\{\!\{ d = v \cdot \mathsf{sum}(L) \}\!\}$

a) Fill in the remaining assertions for each of the code blocks. Determine what must be true at the top of the loop during each iteration to derive P_1's, and use **backward reasoning** to derive Q_0 and Q_1's. (Hint: If you're having trouble finding a starting point for backward reasoning, try to first figure what must be true at the bottom of the loop for each iteration?)

b) The difference in the loop implementations is the order of multiplying and summing of the elements. The first multiplies the elements first and then sums, while the bottom sums and then multiplies. We claim that these are equivalent, but need to prove such a claim first.
Prove that $\text{sum}(\text{mult}(x, L)) = x \cdot \text{sum}(L)$ **by induction** on List L.