Practice Problems Set 13

Fall 25

1. Floyd-Hoare Verification

   The following program purports to compute the factorial of a nonnegative integer `n`. Prove the program's partial correctness (i.e. that if it halts, it computes the factorial of `n`, for any nonnegative input `n`), by giving a Floyd-style proof. Do this by giving an inductive invariant at every point in the program.

```
int r, t;
r = 1;
t = n;
while (t > 0) {
    r = r * t;
    t = t - 1;
}
return r;
```

Solution:

We first construct a Hoare triple for the whole program by annotating the pre- and post-conditions, namely $n \geq 0$ and $r = n!$. Then we annotates the program with inductive invariant at every point in the program to form a Floyd-style proof:

```
{Pre : n ≥ 0}
int r, t;
r = 1;
{n ≥ 0 ∧ r = 1}
t = n;
{n ≥ 0 ∧ n! = r × t! ∧ 0 ≤ t ≤ n}
while (t > 0) {
    r = r * t;
    {n ≥ 0 ∧ n! = r × (t − 1)! ∧ 0 < t ≤ n}
    t = t - 1;
}
return r;
{Post : r = n!}
```

2. Termination

   Consider the following program:

```
int a,b;
a = 1000; b = 0;
while (a != 0 || b != 0) {
    if (b == 0) {
        a = a-1;
        b = f();
    }
    else
        b = b-1;
}
```

In the above, f() is a function that returns arbitrary positive numbers each time it is called (it need not return the same number on two successive invocations). Think of f() as, say, a function that returns a number from the environment (input). More formally, all that we know is that f() returns a value greater than 0.

Prove that the above code terminates always by giving a proof based on ranking functions.

Solution:

We use the ranking function $V(a, b) = \langle a, b\rangle$ with lexicographical order. The annotated program below shows the rank decreases in both branches:

```
while (a != 0  b != 0) {
    decreases ⟨a, b⟩
    // Capture rank at loop entry: V_old = ⟨a, b⟩
    if (b == 0) {
        a = a-1;
        b = f();
        // New state: ⟨a − 1, f()⟩
        // Decrease: ⟨a − 1, f()⟩ ≺_lex ⟨a, 0⟩
        // (Decreases because first component a became smaller)
    }
    else {
        b = b-1;
        // New state: (a, b − 1)
        // Decrease: ⟨a, b − 1⟩ ≺_lex ⟨a, b⟩
        // (Decreases because first component equal, second smaller)
    }
}
```