Problem Set 2

Spring 18

Due: Friday, March 30th, 11:59pm.

1. Floyd-Hoare Style Proofs

   The following program purports to compute the factorial of a nonnegative integer n. Prove the programs 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. Also, give the set of all individual *verification conditions* (which must be valid, of course) that proves that the invariants are really invariant. You can use either weakest pre-conditions or strongest post-conditions to formulate the verification conditions.

   Finally, write down a proof of termination, assuming the pre-condition on the input that $n > 0$, by giving appropriate ranking functions that map states to a well-founded ordering, and argue why this proves termination.

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

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.

3. Syntax-Guided Synthesis

For this problem, you will be synthesizing and analyzing strategies to solve the following problem.

Suppose you have two copiers in a building copier[0] and copier[1]. Each copier can hold capacity units of paper. Every morning, you can go to one of the copiers and add up to maxfill units of paper to it, but you don't have time to go to both of them, so you have to chose which one to go to. However, the printers are online, so before choosing one, you can look online to see how much paper each printer has left in it. Finally, because of printing quotas, you know that the maximum amount of paper used on both printers on a given day is maxuse units.

Your goal is to find a strategy that guarantees that none of the copiers ever runs out of paper. For all the exercises you should use as a staring point the definitions provided in copier.sk. You should submit one sketch for each question below.

(a) Let maxfill=27, capacity=27 and maxuse=13. Find a strategy within the space of programs shown below that guarantees that the copiers never run out of paper *within a bounded horizon of 5 steps*. We model the strategy as a boolean predicate on the state. If true, we refill copier 0; if false, we refill copier 1.

```
pred ::= exp+?? > exp+?? | !pred | pred || pred | pred && pred
exp := copier[??] | step()%7 | 0
```

You will find it useful to restrict the range of ?? to a smaller number of bits. If you use the syntax ??(N), this indicates an N-bit unknown. This notation, however, cannot be used inside a regexp generator ||.

(b) Now we want to see if this generalizes to an unbounded number of steps. In order to do that, we need to characterize the set of safe states. Use the same space of predicates above to find a predicate that characterizes the set of safe states.

(c) Let maxfill = 27, capacity=27 and maxuse=10. Suppose that every Wednesday (day()%7==2) the quotas are extended so that in addition to the maxuse units of paper users are allowed to use, they can collectively use an additional 8 units from copier 0, and every Saturday (day()%7==5), they can use an additional 8 units from copier 1. Find a strategy within the space of programs shown earlier that guarantees that the copiers never run out of paper *within a bounded horizon of 14 steps (two weeks).*

(d) Does your solution from problem 1 satisfy $X\ G\ F\ full_0$, where $full_i$ means that copier i has 'capacity' units of paper? Does your solution from problem 2 satisfy this? Explain.

Hint 1: If you make the number of steps a parameter to the harness, the synthesizer will first look for counterexamples with short horizon, and will only consider the full 14 steps when it is close to a correct solution, improving efficiency significantly. You can use assume to ensure that this input number of steps is less than or equal to 14.

Hint 2: You need a relatively high depth in order to synthesize this from scratch from the grammar above (I used 4, and it takes about 35 min). However, if you explicitly tell the system that your strategy will be of the form

```
if(i%7 < ??){
    return pred (...);
} else {
    return pred (...);
}
```

you can have much smaller bounds for those preds and it will find a solution much faster (about 3 min).

Hint 3: the default unroll factor for loops in sketch is 8, so you will need to explicitly override that in the command line to get a solution.