Assignment 6: Procedures
15-414/15-424 Bug Catching: Automated Program Verification

Due: **11:59pm**, Thursday 11/2/17
Total Points: 50

1. **Context is everything (10 points).** Consider the following rule for dealing with procedure contracts.

$$([\text{callc}])\ \frac{\Gamma \vdash A, \Delta \quad A \vdash [\mathtt{m}()]B \quad \Gamma, B \vdash P}{\Gamma \vdash [\mathtt{m}()]P, \Delta}$$

   Is this rule sound? If so, prove it either by derivation or by giving a semantical argument. It it is not sound, then provide a counterexample proof that is uses this rule, but is unsound (i.e., comes to a false conclusion).

2. **Flawed arguments (10 points).** Consider the following syntax for procedure calls with arguments.

$$\mathtt{m}(e_1, \ldots, e_n)$$

   In the call $\mathtt{m}(e_1, \ldots, e_n)$, the $e_1, \ldots, e_n$ are called the *actual parameters*. For the corresponding declaration,

```
proc m(x1,...,xn) {...}
```

   the $x_1, \ldots, x_n$ are called the *formal parameters*. The actual parameters are terms that are evaluated in the calling context, using the current state at the moment the call is made. The formal parameters are variables that are assigned the corresponding values of the actuals, for later use in the procedure body. When the procedure finishes, the values stored in variables with the same name as the formal parameters are restored to their contents before the call. For example, if we defined the factorial procedure as taking a single argument x:

```
proc fact(x) { if(x = 0) { y:= 1 } else { fact(x-1); y:=y*x; } }
```

   Then we would expect the following formula to be valid: $[x := 42; \mathtt{fact}(100)](x = 42 \land y = 100!)$. Assume for the sake of simplicity that we don't allow recursive procedures in our language, and suppose that we define the semantics of such a procedure call as:

$$[\![\mathtt{m}(e_1, \ldots, e_n)]\!] = [\![v_1 := x_1; \ldots; v_n := x_n; x_1 := e_1; \ldots; x_n := e_n; \alpha; x_1 := v_1; \ldots; x_n := v_n]\!] \qquad (1)$$

   In the above, α is the body of $\mathtt{m}$ and $v_1, \ldots, v_n$ are fresh variables. What is wrong with the semantics shown in (1)? Explain how this definition is different from what we expect in a normal language; if it helps clarify your answer, feel free to give a "counterexample" program that demonstrates the flaw.

3. **Mutual correctness (12 points).** In lecture we discussed the $\langle\text{rec}\rangle$ rule for reasoning about recursive programs, presented below in a slightly simplified form.

$$(\langle\text{rec}\rangle)\ \frac{(\forall \bar{x}. A \land \varphi < n) \to \langle \mathtt{m}() \rangle B \vdash \forall \bar{x}. (A \land \varphi = n) \to \langle \alpha \rangle B \quad A \vdash \varphi \geqslant 0}{\vdash A \to \langle \mathtt{m}() \rangle B} \quad (n \text{ fresh})$$

   Oftentimes programs rely on some form of *mutual recursion*. For example, consider the following (very inefficient) procedures for determining whether a number is even or odd. The input argument is stored in x, and the return variable is y which the value 1 if x is even (respectively, odd), and 0 otherwise.

```
proc even() {
  if(x = 0) { y := 1; } else { x := x - 1; odd() }
}
proc odd() {
  if(x = 0) { y := 0; } else { x := x - 1; even() }
}
```

Modify $\langle \text{rec} \rangle$ to provide a proof rule that supports two mutually-recursive procedures. You do not need to give a proof that your rule is sound, but you should give a concise informal argument for why it is correct. *Hint: Make sure that your rule is able to account for procedures that call themselves, in addition to at most one other mutually-recursive procedure.*

4. **Even and odd contracts (5 points).** Write contracts for the intended behavior of `even` and `odd` from problem 3. That is, you should prove a precondition, postcondition pair for `even` (resp. `odd`) such that if and only if the input x is even (resp. odd), the variable y will contain the value 1 after terminating. Your contracts should specify total correctness. *Note: it is fine if your preconditions specify that the procedures only work on a subset of the integers.*

5. **Prove it (13 points).** Use your rule from problem 3 and your contracts from problem 4 to give a sequent calculus proof that the implementation of `isodd` is correct.