Practice Problems for Final Exam
15-414/15-424 Bug Catching: Automated Program Verification

1. **Sum and max (10 points)** Let α correspond to the following program:

```
while(i < n) {
  if(m < a(i)) {
    m := a(i);
  }
  s := s + a(i);
  i := i + 1;
}
```

   Use the axioms of dynamic logic to conduct a sequent calculus proof that the following formula is valid:

$$s = 0, m = 0, i = 0 \vdash [\alpha] s \leqslant N \cdot m$$

   Be sure to clearly state your loop invariant to help your graders understand your solution.

2. Using the read-over-write axioms, conduct a proof in the sequent calculus of the following formula.

$$a\{i \mapsto e\}(j) = e \to i = j \vee a[j] = e$$

3. In lecture we discussed an axiom for reasoning about array updates:

$$([:=]_{()}) \quad [a(e) := \tilde{e}]p(a) \leftrightarrow p(a\{e \mapsto \tilde{e}\})$$

   Use the following semantics of array update to prove the validity of the formula above:

$$[\![a(e) := \tilde{e}]\!] = \{(\omega, \nu) : \omega = \nu \text{ except } \nu(a) = \omega(a)\{\omega[\![e]\!] \mapsto \omega[\![\tilde{e}]\!]\}$$

4. Consider the following rule for dealing with procedure contracts.

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

   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).

5. 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(x_1,...,x_n) {...}
```

   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: $[x := 42; \texttt{fact}(100)](x = 42 \wedge y = 100!)$. 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:

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

In the above, α is the body of `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.

6. Ghost state is a useful tool in proofs. One of the uses that we discussed in lecture applied to procedure contracts that refer to arguments. Consider the procedure `swap`, with the following contract:

$$(a = x \wedge b = y) \rightarrow [\texttt{swap}(a, b)](a = y \wedge b = x)$$

Suppose that we wanted to use this contract to prove the the following:

$$i \leqslant j \vdash [\texttt{swap}(i, j)](j \leqslant i)$$

Demonstrate how ghost state is used in conjunction with the [callc] rule by conducting a sequent calculus proof of the sequent above. You can assume that `swap` has already been shown to satisfy its contract.

7. Show that the ghost-state introduction rule can derived from the axioms of first-order dynamic logic by giving a sequent calculus proof.

8. Describe in words the role of nondeterminism in verifying data structure invariants. Is it necessary, or would it be possible to perform a convincing verification without it? If it is helpful in formulating your answer, feel free to give an example using the "bit" data structure from Lecture 12.

9. Consider the following data structure, which encapsulates a simple counter that can either be incremented or tested against a threshold. The data itself is represented by an integer.

```
type counter                 = int;
counter init()               { return 0; }
counter inc(counter c)       { return c+1; }
bool test(counter c, int t)  { if(c <= t) then return false else return true; }
```

Because we intend to use this data structure to count real events, which can only happen zero or more but never a negative number of times, we wish to verify that it satisfies the following simple invariant.

$$J \equiv 0 \leqslant c$$

Write an operational model for this data structure, and then explain how the model can be used to verify the invariant using axioms and rules of first-order dynamic logic. The most unambiguous way to answer this question is to provide a DL formula whose validity implies satisfaction of the invariant.

10. Consider an elevator system that services $N > 0$ floors numbered 0 through $N-1$. There is an elevator door at each floor with a call button and an indicator that signals whether or not the elevator has been called. In the elevator cabin there are N destination buttons (one per floor) and N lights that indicate to which floors the elevator has been sent. Present a set of atomic propositions (to make the rest of this problem easier, try to minimize this set!) that are needed to describe the below properties of the system. Then give a CTL formula that specifies each property. To simplify the problem, you can assume that $N = 4$.

(a) A door is never open if the cabin is not present at the given floor.

(b) The indicator lights correctly reflect the current requests: each time a button is pressed, there is a corresponding request that needs to be maintained until fulfillment (if ever).

(c) The elevator only services the requested floors and stands still when there is no request.

(d) All requests are eventually satisfied.

11. Write the CTL operators **EF** , **AG** , **AF** , **A**(· **U** ·) using only **EX** , **EG** , **E**(· **U** ·)

12. Let $P = \{c_1, c_2, n_1, n_2\}$ denote the propositions that P_1 and P_2 are respectively in their critical and non-critical sections. Draw a Büchi automata that formalize the following properties:

    (a) Both processes are never in their critical section at the same time

    (b) Both processes are infinitely often in their critical sections

    (c) The processes alternate entering their critical sections infinitely often