EECS 755 - Final Exam

Fall Semester 2025

December 5, 2025

Exercise 1 *Circle the numbers associated with statements in the following list that are true. (1pt each)*

1. *The* `destruct` *tactic implements proof-by-cases.*
2. *The* `Inductive` *construct defines the set of all elements of a type.*
3. *The* `Definition` *construct can define a recursive function.*
4. *A non-terminating tactical introduces an inconsistency in Coq.*
5. *A tactical takes tactics as arguments.*
6. *$t_1; t_2$ is a tactical that applies t_1 and applies t_2 to the first goal resulting from t_1.*
7. `try` *t attempts tactic t and if it fails returns the proof goal to its previous state.*
8. `apply L in H` *where* `H` *is a hypothesis implements forward reasoning while* `apply L` *implements backward reasoning*
9. `not A` *unfolds to* `A -> False`
10. `split` *takes a goal of the form* `A<->B` *and produces two separate goals* `A->B` *and* `B->A`.
11. *In The Adventures of Buckaroo Bonzai John Lithgow plays a mad scientist trying to return to the 8th dimension using a device called an overthruster.*

Exercise 2 *Using an* Inductive *construction we will define a simple* stack *data structure. (3pts each)*

1. *Using* Inductive *define a stack that can contain any single type. You should minimally define an empty stack and a push operation.*

2. *Define top and pop over your new data structure.*

3. *How would you prove that the top of a stack resulting from pushing x is always x?*

4. *How would you prove that pushing never results in an empty stack?*

5. *What is the difference between the definition of* stack *and the definition of* nat*? Specifically, could you use one to implement the other?*

Exercise 3 *Assume the following AST for a simple language of Booleans and if: (3pts each)*

```
Inductive L : Type :=
| ttrue : L
| tfalse : L
| tand : L -> L -> L
| tnot : L -> L
| tif : L -> L -> L -> L.
```

1. *Define an evaluation function, teval, for L that produces a value of type bool.*

2. *Define an evaluation relation, tevalR, for L that relates an expression to a value of type bool.*

3. *Assuming tif is a traditional if-expression, write a relation that describes an optimization that replaces (tif l1 l2 l2) with l2.*

4. *Assuming tif is a traditional if-expression, write an optimization function that replaces (tif l1 l2 l2) with l2.*

5. *What theorem would you prove to show the optimization relation is correct?*

6. *What theorem would you prove to show the optimization function is correct assuming the optimization relation is correct?*

7. *We want to prove* (forall t1 t2:L, (teval (tif t1 t2 t2)) = (teval t2)). *Should we start with* induction *or* destruct *or something else? (Not* intros.*)*

8. *We want to prove* (forall t1, teval (tand t1 tfalse)) = false. *Should we start with* induction *or* destruct *or something else? (Not* intros.*)*

9. *We want to prove* (forall t1, teval (tand t1 tfalse)) = false. *Now tell me what* intros *does to the theorem.*

10. *We have now proven* Theorem T: (forall t1, teval (tand t1 tfalse)) = false. *Can we use the resulting theorem* T *with either* apply *or* rewrite*?*

Exercise 4 *The definition of IMP from class defines state as a* Map *of variable names to their values with functions to update state and retrieve values from state. Evaluators for IMP accept and produce states. Answer the following about IMP and state: (3pts each)*

1. *When reasoning about some program with* c1 ; c2 *in the assumptions of the proof, would using* destruct *be a good option? If so, are we reasoning forwards or backwards?*

2. *When reasoning about some program with* c1 ; c2 *in the goal of the proof, would using* apply E_Seq *be a good option? If so, are we reasoning forwards or backwards?*

3. *Assume that we've defined a new, swanky optimizer for IMP called* swanky *that inputs an IMP program and returns an optimized IMP program. What theorem would you prove to show the optimizer is correct?*

4. *Our definition of IMP defined three sub-languages -* aexp, bexp, *and* com. *The interpreters for* aexp *and* bexp *accept a term and produce a value. What does the interpreter for* com *accept and produce?*

5. *We say that a variable is* invariant *over program execution if it's value does not change. If we wanted to verify that a variables* X *and* Y *are invariant during IMP program execxution what would we prove? (You may assume the existence of* comR, *a relational definition for* com *evaluation.)*