CIS 5000: Software Foundations **Quiz 5** October 15, 2025

Name (printed): ______________________________ Penn ID (8-digit #): ________________

Please write your name as it is displayed on Gradescope. Indicate your answers with a cross mark ☒. To correct a mistake, fill in the incorrect box (like this: ■) and make a cross in the correct box.

$\boxed{1}$ What does the following expression evaluate to?

```
aeval (APlus (ANum 3) (AMinus (ANum 4) (ANum 1)))
```

☐ true ☐ false ☐ 2 ☐ 7 ☐ 6

$\boxed{2}$ If we know `(APlus e1 e2) ==> n`, which of these can we deduce by inversion alone (mark all that apply)?

☐ `e1` and `e2` must be `ANum` constructors

☐ There exist `n1, n2` such that `e1 ==> n1`, `e2 ==> n2`, and `n = n1 + n2`

☐ There exist `n1, n2` such that `e1 ==> n1`, `e2 ==> n2`, and `n1, n2 < n`

☐ `e1 ==> n` or `e2 ==> n`

$\boxed{3}$ For a given expression `e` and value `n` in the `Imp` language, how many different derivation trees can prove `e ==> n`?

☐ Exactly one

☐ At most one

☐ Potentially many

☐ Depends on the complexity of `e`

$\boxed{4}$ Is the following proposition provable?

```
forall c st st',
  st =[ skip ; c ]=> st' ->
  st =[ c ]=> st'.
```

☐ Yes

☐ No

5 Is the following proposition provable?

```
forall (b : bexp) (c : com) (st st' : state),
  st =[ if b then c else c end ]=> st' ->
  st =[ c ]=> st'.
```

☐ Yes

☐ No

6 Consider the proposition

```
forall b : bexp,
  (forall st, beval st b = true) ->
  forall (c : com) (st : state),
    ~(exists st', st =[ while b do c end ]=> st').
```

(a) Is the proposition provable in Rocq?

☐ Yes

☐ No

(b) Which of the following correctly states the meaning of this proposition?

☐ "For any boolean expression that is always true, there exists a while loop using that expression as its guard that will never reach a final state."

☐ "If there exists a boolean expression that is always true, then no while loop using that expression as their guard will ever reach a final state."

☐ "For all boolean expressions that are always true, for all while loops using that expression as their guard, there is no final state that the loop can reach."

☐ "For any while loop, if its boolean guard is always true then there exists some final state the loop cannot reach."

For Reference

```
Inductive aexp : Type :=
  | ANum (n : nat)
  | APlus (a1 a2 : aexp)
  | AMinus (a1 a2 : aexp)
  | AMult (a1 a2 : aexp).

Fixpoint aeval (a : aexp) : nat :=
  match a with
  | ANum n => n
  | APlus  a1 a2 => (aeval a1) + (aeval a2)
  | AMinus a1 a2 => (aeval a1) - (aeval a2)
  | AMult  a1 a2 => (aeval a1) * (aeval a2)
  end.

Fixpoint aeval (st : state)
               (a : aexp) : nat :=
  match a with
  | ANum n => n
  | AId x => st x
  | <{a1 + a2}> => (aeval st a1) + (aeval st a2)
  | <{a1 - a2}> => (aeval st a1) - (aeval st a2)
  | <{a1 * a2}> => (aeval st a1) * (aeval st a2)
  end.

Inductive aevalR : aexp -> nat -> Prop :=
  | E_ANum (n : nat) :
      aevalR (ANum n) n
  | E_APlus (e1 e2 : aexp) (n1 n2 : nat) :
      aevalR e1 n1 ->
      aevalR e2 n2 ->
      aevalR (APlus e1 e2) (n1 + n2)
  | E_AMinus (e1 e2 : aexp) (n1 n2 : nat) :
      aevalR e1 n1 ->
      aevalR e2 n2 ->
      aevalR (AMinus e1 e2) (n1 - n2)
  | E_AMult (e1 e2 : aexp) (n1 n2 : nat) :
      aevalR e1 n1 ->
      aevalR e2 n2 ->
      aevalR (AMult e1 e2) (n1 * n2).

Definition state := total_map nat.

Inductive com : Type :=
  | CSkip
  | CAsgn (x : string) (a : aexp)
  | CSeq (c1 c2 : com)
  | CIf (b : bexp) (c1 c2 : com)
  | CWhile (b : bexp) (c : com).
```

```
Inductive bexp : Type :=
  | BTrue
  | BFalse
  | BEq (a1 a2 : aexp)
  | BNeq (a1 a2 : aexp)
  | BLe (a1 a2 : aexp)
  | BGt (a1 a2 : aexp)
  | BNot (b : bexp)
  | BAnd (b1 b2 : bexp).

Fixpoint beval (st : state)
               (b : bexp) : bool :=
  match b with
  | <{true}>      => true
  | <{false}>     => false
  | <{a1 = a2}>   => (aeval st a1) =? (aeval st a2)
  | <{a1 <> a2}>  => negb ((aeval st a1) =? (aeval st a2))
  | <{a1 <= a2}>  => (aeval st a1) <=? (aeval st a2)
  | <{a1 > a2}>   => negb ((aeval st a1) <=? (aeval st a2))
  | <{~ b1}>      => negb (beval st b1)
  | <{b1 && b2}>  => andb (beval st b1) (beval st b2)
  end.
```