CSE 230 (Wi 25) Handout

Jan 14, 2025

Question 1

Recall the definition of `Nat` from lecture

```
inductive Nat where
  | zero : Nat
  | succ : Nat -> Nat
  deriving Repr
```

Fill in the blanks below to write a *tail-recursive* version of `add` that adds two `Nat`s.

```
def add_tr(xs ys : Nat) : Nat :=

  ____________________________________________________________

  ____________________________________________________________

  ____________________________________________________________
```

Question 2

Recall the definitions of `sum_list` and `sum_list'` from lecture.

```
def sum_list (xs : List Nat) : Nat :=
  match xs with
  | [] => 0
  | x ::xs' => x + sum_list xs'

def sum_list_tr (xs : List Nat) (acc : Nat): Nat :=
  match xs with
  | [] => acc
  | x :: ys => sum_list_tr ys (acc + x)

def sum_list' (xs: List Nat) := sum_list_tr xs 0
```

Fill in the blanks below to write a generalized induction hypothesis that would let us complete the proof of `sum_list_eq_sum_list'`

```
theorem generalized_ih : _______________________________________________ := by
  /- ignore -/
```

Question 3

Recall the definitions of `Aexp` and `eval` and `eval'` from lecture.

```
inductive Aexp : Type where
  | const : Nat -> Aexp
  | plus  : Aexp -> Aexp -> Aexp
  deriving Repr

def eval (e: Aexp) : Nat :=
  match e with
  | const n => n
  | plus e1 e2 => eval e1 + eval e2

def eval_acc (e: Aexp) (acc: Nat) : Nat :=
  match e with
  | const n => n + acc
  | plus e1 e2 => eval_acc e2 (eval_acc e1 acc)

def eval' (e: Aexp) := eval_acc e 0
```

Fill in the blanks below to write a generalized induction hypothesis that would let us complete the proof of `eval_eq_eval'`

```
theorem generalized_ih : ______________________________________________ := by
  /- ignore -/

theorem eval_eq_eval' (e: Aexp) : eval e = eval' e := by
  intros e
  simp [eval', generalized_ih]
```