

In class we presented the following data-type.

```
data Expr = V String | Add Expr Expr | Let String Expr Expr
```

The structural induction principle for this type is

$$\begin{aligned} & \forall s : \text{String}. P(V s) \wedge \\ & (\forall e1, e2 : \text{Expr}. (P(e1) \wedge P(e2)) \Rightarrow P(\text{Add } e1 \ e2)) \wedge \\ & (\forall e1, e2 : \text{Expr}. (P(e1) \wedge P(e2)) \Rightarrow \forall s : \text{String}. P(\text{Let } s \ e1 \ e2)) \\ & \Rightarrow \forall e : \text{Expr}. P(e) \end{aligned}$$

We defined an eval function which takes an assignment (function mapping strings to integers) and an expression and evaluates it.

```
eval :: (String -> Int) -> Expr -> Int

eval a (V x) = a x
eval a (Add e1 e2) = (eval a e1) + (eval a e2)
eval a (Let x e1 e2) = eval (update (x,eval a e1) a) e2
```

We also defined a point-wise update function which takes an input value pair (x,v) and a function (say f) and returns a function that behaves just like f except that on input x it returns v .

```
update :: (a,b) -> (a -> b) -> a -> b
update (x,v) f = \y -> if x == y then v else f y
```

You may assume the following lemma:

Lemma 0.1.

$$\begin{aligned} & \forall g : \text{String} \rightarrow \text{Int}. \\ & (\forall x : \text{String}. g x \geq 0) \Rightarrow \\ & \forall y : \text{String}. \forall k : \text{Int}. k \geq 0 \Rightarrow \forall z : \text{String}. \text{update } (y, k) g z \geq 0 \end{aligned}$$

Prove the following theorem by induction on the structure of the expression.

Theorem 0.1.

$$\forall e : \text{Expr}. \forall f : \text{String} \rightarrow \text{Int}. (\forall s : \text{String}. f s \geq 0) \Rightarrow \text{eval } f e \geq 0$$