Homework 7

Fletcher Gornick

March 12, 2021

Problem 1 (30 points)

```
type 'a tree = Leaf | Branch of 'a tree * 'a * 'a tree

let rec flip_tree =
  fun t ->
  match t with
  | Leaf -> Leaf
  | Branch (left, root, right) ->
    Branch (flip_tree right, root, flip_tree left)
```

Theorem 1. *For any type* `a` *and any value* `t` *of type* `a tree`*, we have*

`flip_tree (flip_tree t)` = `t`

Domain All values of type `a tree`.

Property $P($`t`$)$: `flip_tree (flip_tree t)` = `t`.

Inductive order $R($`t1`,`t2`$)$: `t1` is a proper subtree of `t2`.

1. `t` = `Leaf`.
   Left-hand side: `flip_tree (flip_tree Leaf)` = `flip_tree Leaf` = `Leaf`.
   Right-hand side: `Leaf`.
   Both sides agree.

2. `t` = `Branch (left, root, right)`.
   Left-hand side:

   `flip_tree (flip_tree (Branch (left, root, right)))`
   = `flip_tree (Branch ((flip_tree right), root, (flip_tree left)))`
   = `Branch (flip_tree (flip_tree left), root, flip_tree (flip_tree right))`

   Right-hand side:

   `Branch (left, root, right)`

   It is thus sufficient to show that

   `flip_tree (flip_tree left)` = `left`
   `flip_tree (flip_tree right)` = `right`

   Which is exactly $P($`left`$)$ and $P($`right`$)$. $P($`left`$)$ and $P($`right`$)$ are implied by the inductive hypothesis because both `left` and `right` are proper subtrees of `t`. That is, $R($`left`,`t`$)$ and $R($`right`,`t`$)$.

Problem 2 (30 points)

```
let rec sum_tree =
  fun t ->
  match t with
  | Leaf -> 0
  | Branch (left, root, right) ->
    sum_tree left + root + sum_tree right
```

Theorem 2. *For any value* `t` *of type* `int tree`*, we have*

`sum_tree (flip_tree t)` = `sum_tree t`

Domain All values of type `int tree`.

Property P(`t`): `sum_tree (flip_tree t)` = `sum_tree t`.

Inductive order R(`t1`,`t2`): `t1` is a proper subtree of `t2`.

1. `t` = `Leaf`.
   Left-hand side: `sum_tree (flip_tree Leaf)` = `sum_tree Leaf` = `0`.
   Right-hand side: `sum_tree Leaf` = `0`.
   Both sides agree.

2. `t` = `Branch (left, root, right)`.
   Left-hand side:

   `sum_tree (flip_tree (Branch (left, root, right)))`
   = `sum_tree (Branch ((flip_tree right), root, (flip_tree left)))`
   = `sum_tree (flip_tree right) + root + sum_tree (flip_tree left)`
   = `sum_tree (flip_tree left) + root + sum_tree (flip_tree right)`

   Right-hand side:

   `sum_tree (Branch (left, root, right))`
   = `sum_tree left + root + sum_tree right`

   It is thus sufficient to show that

   `sum_tree (flip_tree left)` = `sum_tree left`
   `sum_tree (flip_tree right)` = `sum_tree right`

   Which is exactly P(`left`) and P(`right`). P(`left`) and P(`right`) are implied by the inductive hypothesis because both `left` and `right` are proper subtrees of `t`. That is, R(`left`,`t`) and R(`right`,`t`).

Bonus Problem (10 points)

```
let rec length l =
  match l with
  | [] -> 0
  | h :: t -> 1 + length t

let rec size_tree =
  fun t ->
  match t with
  | Leaf -> 0
  | Branch (left, _, right) ->
    size_tree left + 1 + size_tree right

let rec list_of_tree_helper =
  fun t l ->
  match t with
  | Leaf -> l
  | Branch (left, root, right) ->
    list_of_tree_helper left (root :: list_of_tree_helper right l)

let list_of_tree t = list_of_tree_helper t []
```

Lemma 3. *For any type* `a`*, any value* `t` *of type* `a tree`*, and any value* `l` *of type* `a list`*, we have*

`length (list_of_tree_helper t l)` = `size_tree t + length l`

Domain All values of type `a tree` and `a list`.

Property P(`t`): `length (list_of_tree_helper t l)` = `size_tree t + length l`.

Inductive order R(`t1`,`t2`): `t1` is a proper subtree of `t2`.

1. `t` = `Leaf`.
   Left-hand side: `length (list_of_tree_helper Leaf l)` = `length l`.
   Right-hand side: `size_tree Leaf + length l` = `0 + length l` = `length l`.
   Both sides agree.

2. `t` = `Branch (left, root, right)`.
   Left-hand side:

   `length (list_of_tree_helper (Branch (left, root, right)) l)`
   = `length (list_of_tree_helper left (root :: list_of_tree_helper right l))`
   = `size_tree left + length (root :: (list_of_tree_helper right l))`
   = `size_tree left + 1 + length (list_of_tree_helper right l)`
   = `size_tree left + 1 + size_tree right + length l`

   Since the trees `left` and `right` are proper subtrees of `t`, that is, R(`left`,`t`) and R(`right`,`t`), P(`left`) and P(`right`) are implied by the inductive hypothesis. This means

   `length (list_of_tree_helper left)` = `size_tree left + length l`
   `length (list_of_tree_helper right)` = `size_tree right + length l`

Which is why these equalies are made above in *Left-hand side*.

Right-hand side:

`size_tree t + length l`

= `size_tree left + 1 + size_tree right + length l`

Again, both sides agree.

Therefore, by the principle of mathematical induction, the claim `length (list_of_tree_helper t l)` = `size_tree t + length l` must hold for all values `t` and `l` of type `a tree` and `a list` respectively.

Theorem 4. *For any type* `a` *and any value* `t` *of type* `a tree`*, we have*

`length (list_of_tree t)` = `size_tree t`

Domain All values of type `a tree`.

Property $P($`t`$)$: `length (list_of_tree t)` = `size_tree t`.

The function `length (list_of_tree t)` can be rewritten as `length (list_of_tree_helper t [])`. As proven in *Lemma 3*, `length (list_of_tree_helper t l)` = `size_tree t + length l`, so we can apply this to our new case, giving us `length (list_of_tree_helper t [])` = `size_tree t + length []`. Finally, since `length []` = `0`, we have `length (list_of_tree_helper t [])` = `size_tree t + 0` = `size_tree t`.

Again, since `length (list_of_tree_helper t [])` can be rewritten as `length (list_of_tree t)`, we can conclude that `length (list_of_tree t)` = `size_tree t`.