[SPOILERS] Practice Problems for Section 3 - Tests

Subscribe for email updates. PINNED

No tags yet. + Add Tag

Pavel Lepin COMMUNITY TA · a month ago

```
(* All tests should evaluate to true. *)

(** High-Order Fun **)

(* There Can Be Only One *)

val test_fold_map_1 = fold_map (fn x => x + 1) [1, 2, 3, 4, 5] = [2, 3, 4, 5, 6]
val test_fold_map_2 = fold_map (fn x => x) [] = []
val test_fold_map_3 = fold_map not [true, false, true, false] = [false, true, false, t
rue]
val test_fold_map_4 = fold_map (fn x => "fnord " ^ x) ["a", "quick", "brown", "fox"] =
 ["fnord a", "fnord quick", "fnord brown", "fnord fox"]

val test_fold_filter_1 = fold_filter (fn x => x mod 2 = 0) [1, 2, 3, 4, 5] = [2, 4]
val test_fold_filter_2 = fold_filter (fn x => x) [] = []
val test_fold_filter_3 = fold_filter not [true, false, true, false] = [false, false]
val test_fold_filter_4 = fold_filter (fn x => String.size x < 4) ["a", "quick", "brown
", "fox"] = ["a", "fox"]

(* The Evil Twin *)
val test_unfold_1 = unfold (fn x => if x > 3 then NONE else SOME (x + 1, x)) 0 = [0, 1
, 2, 3]
val test_unfold_2 = unfold (fn _ => NONE) false = []
val test_unfold_3 = unfold (fn str => if String.size str > 12 then NONE else SOME ("Ba
nana" ^ str, str)) "" = ["", "Banana", "BananaBanana"]

(* A Novel Approach *)
val test_factorial_1 = factorial 4 = 24
val test_factorial_2 = factorial 0 = 1
val test_factorial_3 = factorial 5 = 120
val test_factorial_4 = factorial 7 = 5040
```

```
(* Unforeseen Developments *)
val test_unfold_map_1 = unfold_map (fn x => x + 1) [1, 2, 3, 4, 5] = [2, 3, 4, 5, 6]
val test_unfold_map_2 = unfold_map (fn x => x) [] = []
val test_unfold_map_3 = unfold_map not [true, false, true, false] = [false, true, fals
e, true]
val test_unfold_map_4 = unfold_map (fn x => "fnord " ^ x) ["a", "quick", "brown", "fox
"] = ["fnord a", "fnord quick", "fnord brown", "fnord fox"]

(* So Imperative *)
val test_do_until_1 = do_until (fn x => x div 2) (fn x => x mod 2 <> 0) 48 = 3
val test_do_until_2 = do_until (fn x => x div 2) (fn x => x mod 2 <> 0) 9 = 9
val test_do_until_3 = do_until (fn x => x ^ " ") (fn x => String.size x > 9) "abcde" =
 "abcde     "

(* Yet Another Factorial *)
val test_imp_factorial_1 = imp_factorial 4 = 24
val test_imp_factorial_2 = imp_factorial 0 = 1
val test_imp_factorial_3 = imp_factorial 5 = 120
val test_imp_factorial_4 = imp_factorial 7 = 5040

(* Fixed Point *)
val test_fixed_point_1 = fixed_point (fn x => x div 2) 17 = 0
val test_fixed_point_2 = fixed_point (fn x => x) 17 = 17

(* Newton's Method *)
val test_my_sqrt_1 = abs (my_sqrt 2.0 - Math.sqrt 2.0) < 0.01
val test_my_sqrt_2 = abs (my_sqrt 9.0 - Math.sqrt 9.0) < 0.01
val test_my_sqrt_3 = abs (my_sqrt 81.0 - Math.sqrt 81.0) < 0.01
val test_my_sqrt_4 = abs (my_sqrt 10.0 - Math.sqrt 10.0) < 0.01
val test_my_sqrt_5 = abs (my_sqrt 3.0 - Math.sqrt 3.0) < 0.01
val test_my_sqrt_6 = abs (my_sqrt 0.25 - Math.sqrt 0.25) < 0.01

(* Deeper Into The Woods *)

val test_tree_fold_1 = tree_fold (fn (l, v, r) => l ^ v ^ r) "!" (node { value = "foo"
, left = node { value = "bar", left = leaf, right = leaf }, right = node { value = "ba
z", left = leaf, right = leaf }}) = "!bar!foo!baz!"
val test_tree_fold_2 = tree_fold (fn (l, v, r) => l * r * v) 1 leaf = 1
val test_tree_fold_3 = tree_fold (fn (l, v, r) => v - l - r) 1 (node { value = 10, lef
t = node { value = 5, left = leaf, right = leaf }, right = node { value = 3, left = le
af, right = leaf }}) = 6

val test_tree_unfold_1 = tree_unfold (fn x => if x = 0 then NONE else SOME (x - 1, x,
x - 1)) 2 = node { value = 2, left = node { value = 1, left = leaf, right = leaf }, ri
```

```
ght = node { value = 1, left = leaf, right = leaf }}
val test_tree_unfold_2 = tree_unfold (fn x => NONE) NONE = leaf
val test_tree_unfold_3 = tree_unfold (fn x => if x = 0 then NONE else SOME (x div 2, x
, x div 3)) 6 = node { value = 6, left = node { value = 3, left = node { value = 1, le
ft = leaf, right = leaf }, right = node { value = 1, left = leaf, right = leaf } }, ri
ght = node { value = 2, left = node { value = 1, left = leaf, right = leaf }, right =
leaf }}

(** A Grand Challenge **)
val test_infer_type_1 = infer_type (conditional (literal_bool, literal_int, binary_int
_op (literal_int, literal_int))) = type_int
val test_infer_type_2 = infer_type literal_int = type_int
val test_infer_type_3 = infer_type literal_bool = type_bool
val test_infer_type_4 = (infer_type (conditional (literal_bool, literal_int, binary_in
t_op (literal_bool, literal_int))); false) handle TypeError => true = true
val test_infer_type_5 = (infer_type (conditional (literal_bool, literal_bool, binary_i
nt_op (literal_int, literal_int))); false) handle TypeError => true = true
val test_infer_type_6 = (infer_type (conditional (literal_bool, literal_int, compariso
n (literal_int, literal_int))); false) handle TypeError => true = true
val test_infer_type_7 = infer_type (conditional (literal_bool, literal_bool, compariso
n (literal_int, literal_int))) = type_bool
val test_infer_type_8 = infer_type (conditional (literal_bool, conditional (literal_bo
ol, literal_bool, comparison (literal_int, binary_int_op (literal_int, literal_int))),
 comparison (literal_int, literal_int))) = type_bool
val test_infer_type_9 = infer_type (conditional (literal_bool, binary_bool_op (literal
_bool, comparison (literal_int, literal_int)), comparison (literal_int, literal_int)))
 = type_bool
val test_infer_type_10 = infer_type (binary_int_op (literal_int, literal_int)) = type_
int
val test_infer_type_11 = (infer_type (binary_int_op (literal_int, literal_bool)); fals
e) handle TypeError => true = true
val test_infer_type_12 = (infer_type (binary_bool_op (literal_int, literal_bool)); fal
se) handle TypeError => true = true
val test_infer_type_13 = (infer_type (comparison (literal_int, literal_bool)); false)
handle TypeError => true = true
val test_infer_type_14 = infer_type (conditional (binary_bool_op (literal_bool, litera
l_bool), literal_bool, comparison (literal_int, literal_int))) = type_bool
val test_infer_type_15 = infer_type (conditional (comparison (literal_int, literal_int
), literal_bool, comparison (literal_int, literal_int))) = type_bool
val test_infer_type_16 = (infer_type (conditional (binary_int_op (literal_int, literal
_int), literal_bool, comparison (literal_int, literal_int))); false) handle TypeError
=> true = true

(** Back To The Future! 2 **)
```

```
(* GCD -- Final Redux *)
val test_gcd_list_1 = gcd_list [18, 12, 3] = 3
val test_gcd_list_2 = gcd_list [18] = 18
val test_gcd_list_3 = gcd_list [18, 12, 13] = 1
val test_gcd_list_4 = gcd_list [10, 18, 12] = 2
val test_gcd_list_5 = gcd_list [100, 1000, 1] = 1
val test_gcd_list_6 = gcd_list [18, 12, 180, 42] = 6
val test_gcd_list_7 = gcd_list [18, 12] = 6

(* Element Of A List -- Final Redux *)
val test_any_divisible_by_1 = any_divisible_by ([13, 1, 20], 5) = true
val test_any_divisible_by_2 = any_divisible_by ([13, 1, 20], 3) = false
val test_any_divisible_by_3 = any_divisible_by ([], 5) = false
val test_any_divisible_by_4 = any_divisible_by ([13, 1, 20], 13) = true
val test_any_divisible_by_5 = any_divisible_by ([13, 1, 20], 12) = false
val test_any_divisible_by_6 = any_divisible_by ([13, 1, 20], 1) = true

(* Quirky Addition -- Continued -- Final Redux *)
val test_add_all_opt_1 = add_all_opt [SOME 1, NONE, SOME 3] = SOME 4
val test_add_all_opt_2 = add_all_opt [] = NONE
val test_add_all_opt_3 = add_all_opt [NONE, NONE, NONE] = NONE
val test_add_all_opt_4 = add_all_opt [SOME 123] = SOME 123
val test_add_all_opt_5 = add_all_opt [NONE, SOME ~1, NONE, NONE] = SOME ~1

(* Flip Flop -- Final Redux *)
val test_alternate_1 = alternate [1, 2, 3, 4] = ~2
val test_alternate_2 = alternate [] = 0
val test_alternate_3 = alternate [~100] = ~100
val test_alternate_4 = alternate [1, ~2, 3, ~4] = 10
val test_alternate_5 = alternate [~1, 2, ~3, 4] = ~10

(* Minimum/Maximum -- Final Redux *)
val test_min_max_1 = min_max [3, 1, 2, 5, 4] = (1, 5)
val test_min_max_2 = min_max [1] = (1, 1)
val test_min_max_3 = min_max [~1000000, 1, 1, 1, 1000000] = (~1000000, 1000000)

(* Lists And Tuples, Oh My! -- Final Redux *)
val test_unzip_1 = unzip [(1, 2), (3, 4), (5, 6)] = ([1, 3, 5], [2, 4, 6])
(* causes polyEqual warning if unzip has a polymorphic type -- that's ok *)
val test_unzip_2 = unzip [] = ([], [])
val test_unzip_3 = unzip [(123, 321), (321, 123)] = ([123, 321], [321, 123])

(* Lists And Tuples, Oh My! -- Continued (1) -- Final Redux *)
val test_zip_1 = zip ([1, 2, 3], [4, 6]) = [(1, 4), (2, 6)]
val test_zip_2 = zip ([], [4, 6]) = []
```

```
val test_zip_3 = zip ([1, 2, 3], []) = []
val test_zip_4 = zip ([], []) = []
val test_zip_5 = zip ([1, 2], [4, 6, 8]) = [(1, 4), (2, 6)]
val test_zip_6 = zip ([1, 2, 3], [4, 6, 8]) = [(1, 4), (2, 6), (3, 8)]

(* BBCA -- Final Redux *)
val test_repeats_list_1 = repeats_list (["abc", "def", "ghi"], [4, 0, 3]) = ["abc", "a
bc", "abc", "abc", "ghi", "ghi", "ghi"]
(* causes polyEqual warning if repeats_list has a polymorphic type -- that's ok *)
val test_repeats_list_2 = repeats_list ([], []) = []
val test_repeats_list_3 = repeats_list (["a"], [10]) = ["a", "a", "a", "a", "a", "a",
"a", "a", "a", "a"]

(* 38 Cons Cells -- Final Redux *)
val test_length_of_a_list_1 = length_of_a_list [1] = 1
val test_length_of_a_list_2 = length_of_a_list [] = 0
val test_length_of_a_list_3 = length_of_a_list [[], [], [1, 2]] = 3
val test_length_of_a_list_4 = length_of_a_list [(1, "hi"), (2, "there")] = 2
val test_length_of_a_list_5 = length_of_a_list ["a", "quick", "brown", "fox"] = 4

(* Forest For The Trees -- Final Redux *)

val test_tree_height_1 = tree_height (node { value = 0, left = node { value = 0, left
= node { value = 0, left = leaf, right = leaf }, right = leaf }, right = node { value
= 0, left = leaf, right = leaf } }) = 3
val test_tree_height_2 = tree_height leaf = 0
val test_tree_height_3 = tree_height (node { value = "abcde", left = leaf, right = lea
f }) = 1
val test_tree_height_4 = tree_height (node { value = true, left = leaf, right = leaf }
) = 1
val test_tree_height_5 = tree_height (node { value = 0, left = leaf, right = node { va
lue = 0, left = node { value = 0, left = leaf, right = leaf }, right = leaf } }) = 3

val test_sum_tree_1 = sum_tree (node { value = 1, left = node { value = 2, left = node
 { value = 3, left = leaf, right = leaf }, right = leaf }, right = node { value = 4, l
eft = leaf, right = leaf } }) = 10
val test_sum_tree_2 = sum_tree leaf = 0
val test_sum_tree_3 = sum_tree (node { value = 1729, left = leaf, right = leaf }) = 17
29
val test_sum_tree_4 = sum_tree (node { value = 32, left = leaf, right = node { value =
 ~60, left = node { value = 17, left = leaf, right = leaf }, right = leaf } }) = ~11

val test_gardener_1 = gardener (node { value = leave_me_alone, left = node { value = p
rune_me, left = node { value = leave_me_alone, left = leaf, right = leaf }, right = le
af }, right = node { value = leave_me_alone, left = leaf, right = leaf } }) = node { v
```

```
alue = leave_me_alone, left = leaf, right = node { value = leave_me_alone, left = leaf
, right = leaf } }
val test_gardener_2 = gardener leaf = leaf
val test_gardener_3 = gardener (node { value = prune_me, left = node { value = prune_m
e, left = node { value = leave_me_alone, left = leaf, right = leaf }, right = leaf },
right = node { value = leave_me_alone, left = leaf, right = leaf } }) = leaf
val test_gardener_4 = gardener (node { value = leave_me_alone, left = node { value = l
eave_me_alone, left = node { value = leave_me_alone, left = leaf, right = leaf }, righ
t = leaf }, right = node { value = leave_me_alone, left = leaf, right = leaf } }) = no
de { value = leave_me_alone, left = node { value = leave_me_alone, left = node { value
 = leave_me_alone, left = leaf, right = leaf }, right = leaf }, right = node { value =
 leave_me_alone, left = leaf, right = leaf } }
val test_gardener_5 = gardener (node { value = leave_me_alone, left = node { value = l
eave_me_alone, left = node { value = prune_me, left = leaf, right = leaf }, right = le
af }, right = node { value = prune_me, left = leaf, right = leaf } }) = node { value =
 leave_me_alone, left = node { value = leave_me_alone, left = leaf, right = leaf }, ri
ght = leaf }
val test_gardener_6 = gardener (node { value = prune_me, left = leaf, right = leaf })
= leaf
val test_gardener_7 = gardener (node { value = leave_me_alone, left = leaf, right = le
af }) = node { value = leave_me_alone, left = leaf, right = leaf }
val test_gardener_8 = gardener (node { value = leave_me_alone, left = leaf, right = no
de { value = prune_me, left = node { value = prune_me, left = leaf, right = leaf }, ri
ght = leaf } }) = node { value = leave_me_alone, left = leaf, right = leaf }
```