
MP 3 – User Defined Types

CS 421 – Spring 2007

Revision 1.1

Assigned February 14, 2007

Due February 23, 2007 at 23:59

Extension two days (20% penalty)

1 Change Log

1.5 Updated “complete heap” definition for the extra credit questions (**important**)

1.1 Fixed a type inconsistency in the description for problem 6 that implied a type conflict with the OCaml output.

1.0 Initial release

2 Objectives and Background

After completing this MP, you should better understand:

- the option datatype
- user-defined datatypes

3 Problems

Warning: You may not use any library functions whatsoever on this MP, except for problems 9 and 10 and the Extra Credit, where you may use @.

3.1 The option Datatype

This section is intended to familiarize you with the `option` datatype.

1. (5 pts) Write `find_first f lst` that returns `Some x` if `x` is the *first* element `y` in `lst` such that `f y` returns `true`, and `None` if there is no such element.

```
# let rec find_first f lst = ...
val find_first : ('a -> bool) -> 'a list -> 'a option = <fun>
# find_first (fun x -> x < 2) [3;2;1];;
- : int option = Some 1
# find_first (fun x -> false) [3;2;1];;
- : int option = None
```

2. (5 pts) (**must be tail recursive**) Write `find_last f lst` that returns `Some x` if `x` is the *last* element `y` in `lst` such that `f y` returns `true`, and `None` if there is no such element.

```

# let find_last f lst = ...
val find_last : ('a -> bool) -> 'a list -> 'a option = <fun>
# find_last (fun x -> x < 2) [3;2;1];;
- : int option = Some 1
# find_last (fun x -> true) [];;
- : 'a option = None

```

3. (10 pts) Write `find_all f lst` that returns `Some xlst` if `xlst` is a non-empty list containing, in the order in which they appear in `lst`, all elements `y` in `lst` such that `f y` returns `true`. If there are no such `y` in `lst`, then `find_all` returns `None`.

```

# let rec find_all f lst = ...
val find_all : ('a -> bool) -> 'a list -> 'a list option = <fun>
# find_all (fun x -> x < 5) [2;4;6;5;3;1];;
- : int list option = Some [2; 4; 3; 1]
# find_all (fun x -> x < 5) [6;7;8];;
- : int list option = None

```

3.2 User Defined Datatypes

For this exercise, you are given the following datatype in the `Mp3common` module. (You need to have open `Mp3common` on top of your `mp3.ml` file.)

```

type 'a tree = Node of 'a tree * 'a * 'a tree | Empty

```

In `Mp3common`, you are also given a function named `complete`. This is an auxiliary function to make testing easier. From the application `complete r n`, you can get a complete tree whose root node has the value `r` and the last leaf node has the value `n`.

3.3 Folding over the tree Datatype

4. (5 pts) Write `tfold f base t` that folds over the tree `t`. `base` is returned for an empty tree. For a non-empty tree, `f` is applied to the result of folding over the left sub-tree, the value stored at the current node, and the result of folding over the right sub-tree, in that order.

```

# let rec tfold f base t = ...
val tfold : ('a -> 'b -> 'a -> 'a) -> 'a -> 'b Mp3common.tree -> 'a = <fun>

```

5. (5 pts) Write `tfold_left f base t` that folds over the tree `t`. `base` is returned for an empty tree.

For a non-empty tree, the function `f` gets as first argument the result of folding over *all* nodes to the *left* of the current node (not just those in the left sub-tree), and as second argument the value of the current node.

```

# let rec tfold_left f base t = ...
val tfold_left : ('a -> 'b -> 'a) -> 'a -> 'b Mp3common.tree -> 'a = <fun>

```

6. (5 pts) Write `tfold_right f base t` that folds over the tree `t`. `base` is returned for an empty tree. For a non-empty tree, the function `f` gets as first argument the value of the current node, and as second argument the result of folding over *all* nodes to the *right* of the current node (not just those in the right sub-tree).

```
# let rec tfold_right f t base = ...
val tfold_right : ('a -> 'b -> 'b) -> 'a Mp3common.tree -> 'b -> 'b = <fun>
```

7. (5 pts) Write `tsum_base` and `tsum_step` such that `tfold tsum_step tsum_base t` sums the values of the nodes of `t`, an `int tree`.

```
# let tsum_base = ...
val tsum_base : int = ...
# let tsum_step = ...
val tsum_step : int -> int -> int = <fun>
# tfold tsum_step tsum_base (Node (Node (Node (Empty, 4, Empty)),
2, Node (Empty, 5, Empty)), 1, Node (Empty, 3, Empty));;
- : int = 15
```

8. (6 pts)

1. Write `tsum_left_base` and `tsum_left_step` such that `tfold_left tsum_left_step tsum_left_base t` sums the values of the nodes of `t`, an `int tree`.

```
# let tsum_left_base = ...
val tsum_left_base : int = ...
# let tsum_left_step = ...
val tsum_left_step : int -> int -> int = <fun>
# tfold_left tsum_left_step tsum_left_base (Node
(Node (Node (Node (Empty, 6, Empty), 2, Node (Empty, 7, Empty)), 0,
Node (Node (Empty, 8, Empty), 3, Node (Empty, 9, Empty))),
-1, Node (Node (Node (Empty, 10, Empty), 4, Empty), 1,
Node (Empty, 5, Empty))));;
- : int = 54
```

2. Write `tsum_right_base` and `tsum_right_step` such that `tfold_right tsum_right_step t tsum_right_base` sums the values of the nodes of `t`, an `int tree`.

```
# let tsum_right_base = ...
val tsum_right_base : int = ...
# let tsum_right_step = ...
val tsum_right_step : int -> int -> int = <fun>
# tfold_right tsum_right_step (Node(Node (Node (Node
(Empty, 5, Empty), 1, Node (Empty, 6, Empty)), -1,
Node (Node (Empty, 7, Empty), 2, Node (Empty, 8, Empty))),
-2, Node (Node (Node (Empty, 9, Empty), 3, Node
(Empty, 10, Empty)), 0, Node (Empty, 4, Empty)))
tsum_right_base ;;
- : int = 52
```

9. (5 pts) Write `tfind_base` and `tfind_step` `f` such that `tfold tfind_step tfind_base t` returns a list of all values `a` in `t` such that `f a` is true, in the order in which they occur in `t`. **Note: You may use @ for this problem.**

```
# let tfind_base = ...
val tfind_base : 'a list = ...
# let tfind_step f = ...
val tfind_step : ('a -> bool) -> 'a list -> 'a -> 'a list -> 'a list = <fun>
# tfold (tfind_step (fun x -> x < 5)) tfind_base
(Node (Node (Empty, 3, Empty), 1,
  Node (Node (Empty, 5, Empty), 2, Node (Empty, 4, Empty))))
- : int list = [3; 1; 2; 4]
```

10. (8 pts)

1. Write `tfind_left_base` and `tfind_left_step` `f` such that `tfold_left tfind_left_step tfind_left_base t` returns a list of all values `a` in `t` such that `f a` is true, in the order in which they occur in `t`. **Note: You may use @ for this problem.**

```
# let tfind_left_base = ...
val tfind_left_base : 'a list = ...
# let tfind_left_step f = ...
val tfind_left_step : ('a -> bool) -> 'a list -> 'a -> 'a list = <fun>
# tfold_left (tfind_left_step (fun x -> false)) tfind_left_base
(Node (Empty, 1, Empty));;
- : int list = []
```

2. Write `tfind_right_base` and `tfind_right_step` `f` such that `tfold_right tfind_right_step t tfind_right_base` returns a list of all values `a` in `t` such that `f a` is true, in the order in which they occur in `t`. **Note: You may use @ for this problem.**

```
# # let tfind_right_base = ...
val tfind_right_base : 'a list = ...
# let tfind_right_step f = ...
val tfind_right_step : ('a -> bool) -> 'a -> 'a list -> 'a list = <fun>
# tfold_right (tfind_right_step (fun x -> x < 3 || x > 5))
(Node (Node (Node (Node (Empty, 8, Empty), 4, Node (Empty, 9, Empty)), 2,
  Node (Empty, 5, Empty)), 1, Node (Node (Empty, 6, Empty), 3,
  Node (Empty, 7, Empty))))
tfind_right_base;;
- : int list = [8; 9; 2; 1; 6; 7]
```

3.4 General Recursion Over the tree Datatype

Sometimes, `fold` is not good enough. For example, we may need to accumulate information (such as current depth) as we traverse a tree.

11. (15 pts) Write `invert t` that takes in a tree `t` and returns a list $[t_1; \dots; t_n]$ of n trees, where n is the number of leaves in `t`. Tree t_i is `t` with l_i , the i_{th} leaf of `t`, as its root.

The idea is to “hang” the tree by the new root and let gravity rearrange the tree accordingly, without crossing any edges.

A sketch of t_i :

Arbitrarily, l_i 's right subtree remains empty; its left subtree should be made to contain the rest of the tree (to ensure consistency with our solution.)

Now imagine traveling along the path from l_i to the old root. At a given node on this path, let's say you came from the right (resp. left) branch, then the edge to the parent becomes the left (right) branch, and the edge to the left (right) sibling becomes the right (left) branch.

Remember: You may not use library functions, such as @.

Note: Although it is not required, this function can be written in time linear in the size of the tree.

```
# let invert t = ...
val invert : 'a Mp3common.tree -> 'a Mp3common.tree list = <fun>
# invert (Node (Node (Empty, 2, Empty), 1, Node (Empty, 3, Empty)));;
- : int Mp3common.tree list =
[Node (Node (Node (Empty, 3, Empty), 1, Empty), 2, Empty);
 Node (Node (Empty, 1, Node (Empty, 2, Empty)), 3, Empty)]
```

3.5 Extra Credit

Note: You may use @ for the extra credit problems.

Definition The **depth** of a root node is 0, and for any node having depth n , its children have depth $n + 1$.

Definition A tree t is a **complete heap** iff:

1. for any node in t , none of its descendants has a value less than its own.
2. every node must have 0 or 2 children, except for the parent of the rightmost leaf, which may have a single left child.
3. all leaves are at depth n or $n - 1$, with all leaves at depth n occurring before any leaf of depth $n - 1$ in an in-order traversal of t (for some n .)

12. (3 pts) Write function `is_heap t` which returns `true` iff `t` is a complete heap.

```
let is_heap t = ...
- : 'a Mp3common.tree -> bool = <fun>
# is_heap (complete 1 5);;
- : bool = true
```

13. (4 pts) Write function `merge t1 t2` that takes in two complete heaps `t1` and `t2`, and returns a new tree t' such that, for each node in `t1` or `t2`, there exists a unique corresponding node with the same value in t' ; furthermore, t' must be a complete heap.

Note: there are multiple reasonable algorithms that solve this problem and meet the above constraint; some return different trees than others.

```
let merge t1 t2 = ...
- : int Mp3common.tree -> int Mp3common.tree -> int Mp3common.tree = <fun>
# complete 1 10 = merge (complete 1 5) (complete 6 10);;
- : bool = true
```