Homework 5: Patterns in Trees

Homework Due: Monday, October 21 at 10 am.

The goal for Homework 5 is to program with data types and prove some theorems about binary search trees.

1 Lab Steps to Complete

As usual, you are encouraged to help each other to finish the lab part.

Prepare Your HW5 Directory

```
cd ~/csci2041
git clone git@github.umn.edu:umn-csci-2041-f19/hw5.git
cd hw5
./setup
```

1. **Run `./setup` every time you clone it.** Please follow the instructions to fix any error or warning.
2. **Do NOT edit files on GitHub directly.** I need to call the IT if you accidentally reveal your homework.

Exercise 1 [Not Graded]

The first part is to implement a simple type checker for OCaml patterns. We are going to define data types `type_`, `value`, and `pattern` to represent the types, values, and patterns in (a small fragment of) OCaml. Here are some examples:

- The integer type `int` will be represented as `TInt` of type `type_`.
- The tuple type `int option * bool` will be represented as `TTuple [TOption TInt; TBool]` of type `type_`.
- The type of lists of nullary tuples `unit list` will be represented as `TList (TTuple [])` of type `type_`.
- The integral value `1` will be represented as `VNum 1` of type `value`.
- The nullary tuple `()` will be represented as `VTuple []` of type `value`.
- The wildcard pattern `_` will be represented as `PWildcard` of type `pattern`.
- The variable pattern `x` will be represented as `PVar "x"` of type `pattern`..
- The tuple pattern `(true, y, z)` will be represented as `PTuple [PTrue; PVar "y"; PVar "z"]` of type `pattern`..
- The list pattern `1 :: []` will be represented as `PCons (PNum 1, PNil)` of type `pattern`..

Check the code in `hw5.ml` for the full definitons of `type_`, `value`, and `pattern`. Study the functions `all2` and `check_value`, and then implement a function `check_pattern` that checks whether a pattern makes sense for a type. Here are some simple test cases:

```
check_pattern TBool PWildcard = true
(* the wildcard pattern works for any type *)

check_pattern (TTuple []) (PVar "x") = true
(* the variable pattern works for any type, too! *)

check_pattern TBool (PTuple []) = false
(* the pattern `()` does not work for `bool` *)

check_pattern (TOption TBool) (PSome 100) = false
(* the pattern `Some 100` does not work for `bool option` *)
```

Exercise 2 [Not Graded]

In the lecture, Favonia showed a basic property of `insert`: it will give you back a binary search tree. In this exercise, we are going to prove a basic property about another function, `member`. Here is the code:

```
let rec member : int -> int tree -> bool
  = fun x -> fun t ->
    match t with
    | Leaf -> false
    | Branch (l, k, r) ->
      match compare_int x k with
      | Equal -> true
      | Less -> member x l
      | Greater -> member x r
```

We want to say `member` never gives false positives. Here is the mathematical criterion of truthness:

Definition 1 (membership) *For any value* `x` *of type* `int`*, the predicate* Member(`x`, −) *on values of type* `int tree` *is defined inductively as follows:*

- Member(`x`, `Leaf`)*: false.*
- Member(`x`, `Branch (l, k, r)`)*:* Member(`x`, `l`) *or* ⟦`x`⟧ = ⟦`k`⟧ *or* Member(`x`, `r`).

Member(`x`, `e`) *on expressions* `e` *is defined to be* Member(`x`, `v`) *where* `v` *is the value of* `e`.

Please prove the following theorem:

Theorem 1 *For any value* `x` *of type* `int` *and any value* `t` *of type* `int tree`*, if* `member x t = true` *then* Member(`x`, `t`).

Hints: If there's another layer of pattern-matching, the standard approach is to start another layer of induction.

This concludes the lab part of Homework 3.

2 Take-Home Portion

Your code for Exercise 3. The code should go into `hw5.ml`. You *can* use functions from the OCaml standard library, as long as they are available in **OCaml 4.05.0**, and you are using them correctly. However, due to the incompatibility between different versions of OCaml, it is recommended to implement your own helper functions if you are not working on a CSE machine or VOLE. We do not care about those auxiliary declarations, as long as by the end of your program, the names designated in the exercises point to the correct functions.

Your proofs for Exercise 4. You need to typeset or scan your solution to Exercise 4 as one single file `hw5.pdf` in the homework repository, and submit it through `git push`. It is encouraged to use professional typesetting software such as LaTeX and you can reuse the code in `README.tex` for your work. Otherwise, your hand-writing must be clear and legible, and it must be scanned and committed as one PDF named `hw5.pdf`.

Scheduled network down time. There is a scheduled network maintenance in Keller Hall during 6am–8am starting October 21, 2019, which is several hours before the deadline. Please plan accordingly. It is not a valid basis for a deadline extension.

Exercise 3 [2B 30pts]

Implement a function named `match_value` of type `pattern -> value -> (string * value) list option`. The expression

```
match_value p v
```

should return the list of variable bindings induced by matching the value `v` against the pattern `p`. A binding is a pair of the variable name and the value that the variable is bound to. If the matching successes, the function should return `Some l` where `l` is a list of bindings. Otherwise, if the matching fails, the function should return `None`. You may assume the value representation `v` and the pattern representation `p` are of the same type. (See the lab exercise.) You may also assume there are no duplicate variable patterns inside `p`. The order of bindings does not matter; any permutation is correct. However, there should not have duplicate bindings. Here are some simple test cases:

```
match_value PWildcard (VNum 1) = Some []
(* the wildcard pattern does not induce any binding *)

match_value (PVar "x") VTrue = Some [("x", VTrue)]
(* the variable pattern induces a binding *)

match_value (PTuple [PVar "x"; PVar "y"]) (VTuple [VNum 1; VTrue])
  = Some [("x", VNum 1); ("y", VTrue)]

match_value (PSome (PTuple [])) VNone = None
```

The auto-checker will be enabled after 9 pm today (October 15).

Hints: There is an interesting function in `hw5.ml`.

Exercise 4.1 [2B 30pts]

```
(* renamed from `max_tree` to `max` in Homework 5 to reduce your typing *)
let rec max : 'a . 'a tree -> 'a option
  = fun t ->
    match t with
    | Leaf -> None
    | Branch (_, k, r) ->
      match max r with
      | Some m -> Some m
      | None -> Some k
```

Prove this theorem:

Theorem 2 *For any value* `x` *of type* `int` *and any value* `t` *of type* `int tree` *such that* BST(`t`)*, if* `max x t = Some m` *then* MEMBER(`m`, `t`).

Hints: Do the lab exercise first!

Exercise 4.2 [2A 30pts, bonus 2B 10pts]

```
(* This is yet another version of `delete` that was not covered in the lecture.
 * It avoids the impossible case in the longer version and the potential inefficacy
 * in the shorter version. It turns out to be one of the easiest variants to finish
 * the proof, too. *)
```

```
let rec delete : int -> int tree -> int tree
  = fun x -> fun t ->
    match t with
    | Leaf -> Leaf
    | Branch (l, k, r) ->
      match compare_int x k with
      | Less -> Branch (delete x l, k, r)
      | Greater -> Branch (l, k, delete x r)
      | Equal ->
        match r with
        | Leaf -> l
        | _ ->
          match max l with
          | Some m -> Branch (delete m l, m, r)
          | None -> r (* l = Leaf *)
```

Prove the following theorem, which is an important step toward the full correctness of `delete`.

Theorem 3 *For any value* `x` *of type* `int` *and any value* `t` *of type* `int tree` *such that* BST(`t`)*, the following two statements hold:*

- *If* `max x t` = `Some m`*, then* AllLess(`m`, `delete m t`)*.*
- *If* `max x t` = `None`*, then* `t` = `Leaf`*.*

Hints: Follow the code of `max`, and use the information you collect to run the code of `delete`. Review the notes if you are unsure how to work with the predicates AllLess(`i`, −) and AllGreater(`i`, −).

Anonymous Feedback

We are collecting anonymous feedback throughout the semester.

Due

Homework Due: Monday, October 21 at 10 am.