Foundations of Computer Science

Supervision 2

Dima Szamozvancev and Michael Lee

1 More on Lists

Prerequisites: Lecture 4

1. The functions `zip` and `unzip` seem like they are each other's opposites. Which of the following statements are true? Justify your answers.

a) For all `'a`, `'b`, for all `xs` of type `('a * 'b) list`, `zip (unzip xs) = xs`
b) For all `'a`, for all `xs` of type `'a list`, `zip (unzip xs) = xs`
c) For all `'a`, `'b`, for all `xsp` of type `'a list * 'b list`, `unzip (zip xsp) = xsp`

2. Mathematically, two functions $f : A \to B$ and $g : B \to A$ are inverses of each other if for all arguments $a \in A$, and for all arguments $b \in B$

$$g(f(a)) = a \qquad f(g(b)) = b$$

If only the first equation holds, we call g the *left inverse* of f, (g inverts f if on the left of f, gf). If only the second condition holds, we call g the *right inverse* of f (g inverts f if on the right of f, fg).
Let `zip` be f and `unzip` be g.

   a) What are A and B in this context?
   b) Is `unzip` the inverse, right inverse or left inverse of `zip`?

3. a) Use the `member` function on Page 32 to implement the function `inter` that calculates the *intersection* of two OCaml lists – that is, `inter xs ys` returns the list of elements that occur both in `xs` and `ys`.

   b) Similarly, code a function to implement set union. It should avoid introducing repetitions, for example the union of the lists `[4;7;1]` and `[6;4;7]` should be `[1;6;4;7]` (though the order should not matter).

   c) Implement a module with the following interface. Choose the right implementation of `t`!

```
module Set: sig
  type 'a t  (* the type of sets *)
  val empty: 'a t
  val singleton: 'a -> 'a t
  val union: 'a t -> 'a t -> 'a t
  val inter: 'a t -> 'a t -> 'a t
end = struct
  (*Write code here*)
end
```

d) OCaml's implementation of Set requires that two elements of a set can be *compared*: i.e. beyond saying that two elements are equal or not equal, you can also say specify how: x is not equals to **and** less than y. Why might this restriction allow for more efficient implementations for sets? (Hint: you might need to wait for lecture 6)

e) *[Optional]* OCaml's implementation of Set is a **functor**, a function that takes a module, and returns a module. In contrast with our earlier definition of Sets, this allows us to accept not only a type t, but also a custom comparator on the type.

```
module Make:
  functor (Ord : OrderedType) -> S  with type elt = Ord.t
```

For example, we can build an IntSet using

```
module IntSet = Set.Make(struct
  type t = int
  let compare a b = a - b
end)
```

Explain why it might make sense to require the user to pass in a compare function, rather than relying on defaults.

4. Code a function that takes a list of integers and returns two lists, the first consisting of all nonnegative numbers found in the input and the second consisting of all the negative numbers. How would you adapt this function so it can be used to implement a sorting algorithm?

5. How does this version of zip differ from the one in the course?

```
let rec zip = function
  | (x::xs, y::ys) -> (x,y) :: zip (xs,ys)
  | ([], [])       -> []
```

6. Explain how the simple "making change" function in the lecture notes behaves in the following cases:

   a) The till is set to [50, 10, 20, 5, 2, 1]

   b) The till is set to [50, 50, 20, 10, 5, 2, 1]

   c) The till is set to [50, 20, 10, 5, 2, 1, 0]

   d) The till is set to [50, 20, 10, 5, 2, 1, -1]

   e) The amt is set to -1

2 Sorting

Prerequisites: Lecture 5

For certain algorithms, it is important that the list is sorted, i.e. in non-increasing or decreasing order.

This set of questions will help you think about when sorting is necessary, and how to perform sorting when it is required

1. You are given a long list of integers. Would you rather:
   a) Find a single element with linear search or sort the list and use binary search?
   b) Find a lot of elements with linear search or sort the list and use binary search?
   c) Find all duplicates by pairwise comparison or sort the list and check for adjacent values?
2. Another sorting algorithm (*bubble sort*) consists of looking at adjacent pairs of elements, exchanging them if they are out of order and repeating this process until no more exchanges are possible.
   a) Characterise the lists for which bubble sort will have the worst performance (these lists are sometimes known as *pathological*)
   b) Analyse the big O time complexity of this approach.
   c) Suggest a scenario where it might be appropriate to use bubble sort
3. Implement bubble sort in OCaml.
4. Another sorting algorithm (selection sort) consists of looking at the elements to be sorted, identifying and removing a minimal element, which is placed at the head of the result. The tail is obtained by recursively sorting the remaining elements.
   a) State, with justification, the time complexity of this approach.
   b) Implement selection sort using OCaml.
5. Many programming languages, like Python, offer a `sort` function to be used on lists / arrays.
   a) How could you choose what sorting algorithm to use?
   b) How would you let the user override the default choice, if necessary?

3 Trees

Prerequisites: Lecture 6
If lists are one of two fundamental data structures, then trees are the other. This section aims to familiarise you with trees in OCaml.

1. Examine the following function declaration. What does `ftree (1,n)` accomplish?

```
let rec ftree k = function
  | 0 -> Lf
  | n -> Br(k, ftree (2*k) (n-1), ftree (2*k+1) (n-1))
```

2. a) Write an function taking a binary tree labelled with integers and returning their sum.
   b) Write a function taking a binary tree labelled with integers and returning their product.

   c) Compare your answers to parts (a) and (b). Can you re-write them to share the majority of their implementations?

3. A set of elements is *defined inductively* if it is defined using a set of rules. Each rule must take one of two forms:

   i) $\frac{}{x \in X}$, which states that x must be in X
   ii) $\frac{x_1 \in X, x_2 \in X, \ldots x_n \in X}{y \in X}$, which gives a "formula" for forming new elements (y) in the set, from existing elements in the set $(x_1, \ldots, x_n)$

   (To be pedantic, the first form is subsumed by the second form, where $n = 0$, so there is only really one form of rule.) As an example, the set of natural numbers can be defined inductively, as follows:

   i) $\frac{}{0 \in \mathbb{N}}$, which states that 0 is a natural number
   ii) $\frac{n \in \mathbb{N}}{s(n) \in \mathbb{N}}$, which states that, if n is a natural number, then its successor ($s(n)$, or $n+1$) is a natural number.

   Thus, inductively defined sets are defined as a set of base elements (in the case of natural numbers, 0), and additionally a set of rules that form new elements from existing elements.

   Define the set of binary trees of natural numbers inductively (*you do not have to redefine natural numbers*). Compare your answer to the definition of trees as an OCaml datatype, as given in the notes.

4. There is a very natural way to prove properties of a set that is defined inductively. This is known as the **induction principle**. For example, to prove a property Φ about the naturals, we can:

   i) Prove that the property holds for 0
   ii) Prove that if the property holds for n, then it holds for $n+1$.

   Since these are the **only** ways to build natural numbers, we have proved it for all natural numbers.

   Using your inductive definition of trees, derive the induction principle over trees of naturals.

5. In what sense is mathematical induction a special case of structural induction?

6. Prove the inequality involving the depth and size of a binary tree `t` from Page 53.

$$\forall \text{ trees } \mathtt{t}.\ \mathrm{count}(\mathtt{t}) \leq 2^{\mathrm{depth}(\mathtt{t})} - 1$$

   Hint: The easiest way to do this is with *structural induction*
   You can also try standard mathematical induction on some numerical property of the tree, but be careful with that you are assuming and what you are proving!

4 Datatypes

Prerequisites: Lecture 6

OCaml has some built in types, like `int` and `float`. OCaml also allows the user to create **custom**

datatypes. In other words: your own, named, types. Trees are just one example of a datatype: in this section, we will see more such examples.

1. You have seen how binary trees can be defined as a datatype `type 'a tree = Lf | Br of 'a tree * 'a * 'a tree`
   a) Define lists similarly. Use `Nil` for the empty list, and `Cons` for `::`.
   b) Define also *unary* trees
   c) Establish the relationship between lists and unary trees
2. Give the declaration of an OCaml datatype for arithmetic expressions that have the following possible shapes: floats, variables (represented by strings), or expressions of the form

   $$-E \qquad \text{or} \qquad E+E \qquad \text{or} \qquad E \times E$$

   Hint: recall how expressions differ from statements, and how their characteristic structure could be captured as a data type. It's a lot simpler than it may seem!
3. Continuing the previous exercise, write a function `eval : expr -> float` that evaluates an expression. If the expression contains any variables, your function should raise an exception indicating the variable name.
4. Define an `'a option` datatype that describes an optional value. Elements of `'a option` are either `None`, or `Some(v)`, where `v` is an element of `'a`.
5. There are two proposals for representing option types in memory (`None` is represented as some null value, e.g. a null pointer):
   i) Option 1.
   `Some v` is represented exactly the same as `v` (so there would be no difference between `32` and `Some(32)`)
   ii) Option 2.
   `Some v` has a different memory representation to `v`. For example, `Some v` is a pointer to a block of memory that tells you it is `Some(v)`, and then points at the value of `v`.
   Explain why the first option, while more efficient, cannot be used for option types in OCaml.
   Hint: can you think of a value `v` for which it would be really bad if `Some(v) == v`?
   In OCaml, the `unboxed` annotation tells the compiler to use (i) instead of (ii). Why might it make sense to add such annotations to the language?

5 Dictionaries and functional arrays

Prerequisites: Lecture 7
As stated previously, lists and trees are the tools of functional programmers. In this section, we will consider arrays and dictionaries: the tools of imperative programmers. Rather than switch languages, we will see how to encode these representations in OCaml.

1. Draw the binary search tree that arises from successively inserting the following pairs into the empty tree: `(Alice, 6)`, `(Tobias, 2)`, `(Gerald, 8)`, `(Lucy, 9)`. Then repeat this task

using the order (Gerald, 8), (Alice, 6), (Lucy, 9), (Tobias, 2). Why are results different? How could we avoid the issue encountered in the first case?

2. Code an insertion function for binary search trees (with keys of string type, and values of polymorphic 'a type). It should resemble the existing update function except that it should raise the exception Collision if the item to be inserted is already present. Now try modifying your function so that Collision returns the value previously stored in the dictionary at the given key. What problems do you encounter and why?
3. Describe and code an algorithm for deleting an entry from a binary search tree. Comment on the suitability of your approach. There are two reasonable methods – one is simple, the other is efficient but a bit more tricky.
4. Write a function to remove the first element from a functional array. All the other elements are to have their subscripts reduced by one. The cost of this operation should be linear in the size of the array.
5. Show that the functions preorder, inorder and postorder all require $O(n^2)$ time in the worst case, where n is the size of the tree.
6. Show that the functions preord, inord and postord take linear time in the size of the tree.

6 Optional Questions

1. We know nothing about the functions f and g other than their polymorphic types:

```
> val f : 'a * 'b -> 'b * 'a = <fun>
> val g : 'a -> 'a list = <fun>
```

Suppose that f (1, true) and g 0 are evaluated and return their results. State, with reasons, what you think the resulting *values* will be, and how the functions can be defined. Can any of the definitions be changed if "return their results" wasn't a requirement?

2. Give a declaration of the data type day for the days of the week. Comment on the practicality of such a datatype in a calendar application.
3. Write a function day_from_date : int -> int -> int -> day which calculates the day of the week of a date given as integers. For example, day_from_date 2020 10 13 would evaluate to Tuesday (or whatever encoding you used in your definition of day above). *Hint*: Using Zeller's Rule might be the easiest approach.