Foundations of Computer Science – Supervision 1 Supplementary Sheet

Heidi Howard

October 20, 2014

Welcome to your first supervision in Foundations of Computer Science, we will begin by discussing this weeks problem sheet and answering your questions from the lectures. Later we will move onto looking at new questions and topics.

Marking explained

Each question has been given a number 1,2 or 3, the meaning of these values is as follows:

1. This answer is very good and there are no particular issues with it.
2. This answer is ok but could be improved. Often answers will receive a 2 if they have provided a correct but inefficient implementation or they have failed to handle some cases.
3. This answer is not complete or is incorrect, we will discuss in the supervision and another answer to this question should be submitted in next week's work.

The following are some questions that we may refer to in the supervision.

Exercise 1:

An algorithm requires $T(n)$ units of space given an input of size n, where the function T satisfies the recurrence.

$$T(1) = 1$$

and when $(n > 1)$

$$T(n+1) = T(n) + 1 \tag{1}$$

$$T(n+1) = T(n) + n \tag{2}$$

$$T(n) = T(n/2) + 1 \tag{3}$$

Find the algorithms space requirement in big O notations.

Exercise 2:

We know nothing about the function f other than its polymorphic type:

```
val f = fn: 'a -> 'b list
```

Suppose that f 7 is evaluated and returns its result. State, with reasons, what you think the result value will be. [taken from exercise 4.6 from the course notes]

Exercise 3:

Together we are going to implement a basic list library, the interface for our library is as follows:

```
(* hd returns the first element in a non-empty list *)
val hd = fn: 'a list -> 'a

(* given a non-empty list, tl returns the list minus the first element *)
val tl = fn: 'a list -> 'a list

(* null returns true if list is empty, and false otherwise *)
val null = fn: 'a list -> bool

(* max returns the maximum integer from a list of integers,
        if the list is empty then 0 is returned *)
val max = fn: int list -> int

(* converts a real list to a integer list *)
val to_int_list = fn: real list -> int list

(* converts a int list to a real list *)
val to_real_list = fn: int list -> real list

(* counts the number of true's in a list of bools,
        recursive implementation *)
val count_true = fn: bool list -> int

(* counts the number of true's in a list of bools,
        iterative implementation *)
val count_truei = fn: bool list -> int

(* computes the length of a list, recursive implementation *)
val length = fn: 'a list -> int

(* computes the length of a list, iterative implementation *)
val lengthi = fn: 'a list -> int
```

Exercise 4:

Write a function flatten, which given a list of lists, returns a single list with all the same elements. For example: flatten [[1,2],[3,4],[],[7.8]] returns [1,2,3,4,7,8].

```
val flatten = fn: 'a list list -> 'a list
```

How can we use this function to build flatten3 and flatten4:

```
val flatten3 = fn: 'a list list list -> 'a list
val flatten4 = fn: 'a list list list -> 'a list
```

Exercise 5:

This week we wrote to function which takes a list and returns a new list containing only elements at even indexes, e.g. given [a, b, c, d] it should return [b, d]. Which of the following implementations are correct? If they are incorrect say why.

```
fun even [] = []
|       even [x] = []
|       even (_::y::ys) = y :: (even ys)
```

```
fun even [] = []
|       even [x] = []
|       even ((_::y)::ys) = y :: (even ys)
```

```
fun even [] = []
|       even [x] = []
|       even (_::(y::ys)) = y :: (even ys)
```

Exercise 6:

Implement the following ML function:

```
(* given two list, add returns a new list where the
        elements of the two lists have been added together pairwise *)
(* if lists are different length, then the result
        is equivient to padding the shorter list with zeros *)
(* e.g. add_lists ([1,2,3,4],[1,2,3]) will return [2,4,6,4] *)
val add_lists = fn: int list * int list -> int list
```

Exercise 7:

Why use pattern matching syntax instead of it-then-else syntax?

Exercise 8:

Write a function called sorted which returns true or false depending on whether a list is sorted. Decide for yourself how you would handle an empty list.

Exercise 9:

Write a function called dedup which removed duplicates from a sorted list. What is the type of this function? Is the function polymorphic?

Exercise 10:

We can no longer assume that the input to dedup is sorted, but we do know that it contains only one duplicate item and each number between 1 and n-1 (where n is length of the list) will occur at most once. Example inputs are [1,2,2] or [5,3,4,2,1,3]. Write a new function dedup2 which handles this case.