Foundations of Computer Science
Supervision Workbook

Michealmas 2015

Supervisior: Heidi Howard
hh360@cam.ac.uk

Last updated: October 12, 2015

Welcome to our supervision workbook for Foundations of Computer Science in Michaelmas 2015. Read through the following information carefully.

If you find any mistakes in this workbook then please do let me know, I will thank you with appreciation and chocolate. If you have any difficulties or questions, do not hesitate to contact me. I am best contacted by email at `hh360@cam.ac.uk`, I aim to response to student within 24 hours. Otherwise, you can make use of our course mailing list by sending an email to `cl-foundscs-heidi@lists.cam.ac.uk`.

1 Supervision Guidence

1.1 Preperation

There are 4 excellent resources at your disposal to help you to learn the foundations of computer science and programming in ML. Of course there exists many more and if you find any of particular interest, please do send me a link.

1.1.1 Course Notes

The course's webpages can be found at `www.cl.cam.ac.uk`, Current Students -¿ Taught lecture courses and modules -¿ Part IA CST -¿ Foundations of Computer Science. This site includes the details course notes, written by Larry Paulson and slides he will cover during lectures. These notes are an excellent resource and by the end of term, you should be familar with all the material covered. You also have access to all previous year notes, these can be reached from `http://www.cl.cam.ac.uk/teaching/material.html` or by editing the year in the course page URL.

1.1.2 ML Compiler

I highly suggest that you both get ML set up on your personal machine as quickly as possible. There are many flavours of ML, SML is one used in this course and specifically the PolyML compiler. The course webpage includes information on how to install PolyML, as does the PolyML homepage[1].

1.1.3 ML for the working programmer

This is the best book for this course. You should be able to find it in your college library or in the computer lab library, otherwise you usually buy a second hand copy online for around 10. If you do not have access to a copy, then let me know as I have a few spare copies.

1.1.4 Past Tripos Questions

In my opinion, past tripos question are the best way to revise for cambridge tripos questions. Again, you can find them through the course webpages.

[1] `http://www.polyml.org/`

1.2 Timetable

The timetable for your supervisions with me this term is as follows:

Fri 16th Oct 23:30	Supervision 1 deadline (covering L1-L3)
Sun 18th Oct 23:30	Supervision 1 feedback
Mon 19th Oct - Wed 21st Oct	Supervision 1
Fri 23rd Oct 23:30	Supervision 2 deadline (covering L4-L6)
Sun 25th Oct 23:30	Supervision 2 feedback
Mon 26th Oct - Wed 28th Oct	Supervison 2
Fri 30th Oct 23:30	Supervision 3 deadline (covering L7-9)
Sun 1st Nov 23:30	Supervision 3 feedback
Mon 2nd Nov - Wed 4th Nov	Supervison 3
Fri 6th Nov 23:30	Supervison 4 deadline (covering L10-12)
Sun 8th Nov 23:30	Supervison 4 feedback
Mon 9th Nov - Wed 11th Nov	Supervison 4

1.3 Booking a supervision

We will be using Otter, as our supervision booking system. If you have any issues with the booking system then let me know. Please book your supervision as soon as possible. Booking operates on a first come first served basis. If another group has the slot you want, then email them directly and organise to swap, and let me know once a swap have been agreed.

If none of the slots are suitable, then email me as a group and Ill add more slots for you.

Booking is now open here http://otter.cl.cam.ac.uk/signups/events/rek117u7

1.4 Submitting supervision work

Supervision work is due at 11:30 pm on Friday evenings, unless stated otherwise. If you have not finished your work by the deadline, submit it unfinished and then submit the remainder later. I cant guarantee I will mark work submitted after the deadline.

Submit work electronically by email to `hh360@cam.ac.uk`.

Please ensure that the subject field and filename clearly indicates the supervision number, group number and your CRSID, in that order and seperated by underscores. Examples of this format include s2_g6_hh360 or s1_g10_ar123.

The source file submitted should also include the following proforma:

```
Name:
```

```
CRSID:
College:
Supervision Number:
Group Number:
Date/Time of Supervision:
```

There may also be other files such as a scan or photo of some work done on paper or handwritten practice tripos solution when practicing for exams. Please ensure images are of a good quality and there the file sizes are not too large.

Do not send me work as a docx file.

```
(* last takes a list and returns the last item in the list. This function
   doesn't handle empty lists *)
(* val last = fn: 'a list -> 'a *)
fun last [x] = x
|   last (x::xs) = last xs

(* last [3] returns 3 *)
(* last [3,5,7,9] returns 3 *)
```

1.5 Before a supervision

Come prepared for the supervision, ensure that bring at least: your lecture notes as well as paper and writing instruments (or electronic equivalent).

1.6 Marking

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.

1.7 Revision

If you feel it will be useful to you, I will be running revision supervisions at the start of lent and the start of easter term.

2 Problem Sheet One

This problem sheet will introduce you to programming in ML. We will cover writing functions and variables, the if-then-else construct, the difference between recursion and iteration and the bool, int and float types. We will cover the syntax for lists and introduce some common list operations. You will write polymorphic functions and use pattern matching to implement recursion.

The most challenge part for most of you will be recurrence relations, covered in exercises 5,6 and 7. If you are struggling, contact me for a hint.

This problem sheet will cover the material from lectures 1 to 3.

Lecture 1

Exercise 1:

One naive solution to the year 2000 bug involves continuing to storing years as two digits, but interpreting them such that 50 means 1950 and 49 means 2049. Code ML functions to:

(a) convert from this new representation to the classical 4 digit representation and back.

(b) test if one year is greater than another years.

(c) add/subtract some given number of years from another year.

[adapted from exercise 1.1 and 1.2 from the course notes]

Exercise 2:

This lecture introduced two method of representing numerical values: reals and integer. This questions and next explore the relationship between ML functions operating on these different representations. Write a function for finding the median of three integers and write another one for finding the median of three reals. Answer the following questions about your functions.

(a) What are the types of these two function?

(b) Why are the types different?

(c) Do either of your functions use *type constraints*?

(d) If so, why did you use type constraints and where they necessary?

(e) Do you think that ML programmers should include *type constraints* when they are not necessary?

Exercise 3:

Write a function for finding the mean of three integers and write another one for finding the mean of three reals. Answer the following questions about your functions.

(a) What are the types of these two function and why they are different?

(b) Which operators have you used in your function?

(c) Which operators are *overloaded* and which ones are not?

(d) Do you think overloaded operators are useful?

Lecture 2

Exercise 4:
Write a function to compute the factorial of a number. Provide both recursive and iterative versions of your function and comment on the improvement in efficiency.

Exercise 5:
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 $(n > 1)$

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

Prove that the algorithms space requirement in big O notations is linear.

Exercise 6:
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 $(n > 1)$

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

Prove that the algorithms space requirement in big O notations is quadratic.

Exercise 7:
(Hard) 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 $(n > 1)$

$$T(n) = T(n/2) + n$$

Prove that the algorithms space requirement in big O notations is linear.

[adapted for the last part of tripos question 1999 P1 Q5]

Lecture 3

Exercise 8:

Write a function to compute the sum of a list's elements. Provide both recursive and iterative versions of your function and comment on the improvement in efficiency. Is this function polymorphic? If so, then demonstrate it and if not, explain why.

[adapted from exercise 3.1 from the course notes]

Exercise 9:

Write a function called nth, which takes a list and an integer n and returns the nth element of the list. Assume lists are indexed from 0. Is this function polymorphic? If so, then demonstrate it and if not, explain why.

Exercise 10:

Write a 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].

[adapted from exercise 3.3 from the course notes]

Exercise 11:

(Hard) 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]. This type of flatten is as follows:

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

How can we use this function to build functions flatten2 and flatten3:

```
val flatten2 = fn: 'a list list list -> 'a list
val flatten3 = fn: 'a list list list list -> 'a list
```

3 Problem Sheet Two

This supervision is all about lists, sorting and datatypes. You should come to this supervision prepared to explain the making change and quicksort examples for the course notes.

This supervision will cover the material from lectures 4 to 6.

Lecture 4

Exercise 1:
Write a function which takes two list and return true if the first list is present anywhere within the second and false otherwise. For example `sublist ([1,2],[0,1,2,3])` returns true but `sublist ([3,5],[5,4,3,2,1])` returns false.

Exercise 2:
Write a function to test if a list is a palindrome. You may make use of functions in the lecture course so far.

A palindrome is a word (or here, for the sake of exercising, a list of elements) that is the same when it's read in both ways (i.e., from left to right or from right to left). Since we're using lists to represent arbitrary words, any list with a single element is a palindrome, and an empty list is also a palindrome.

Exercise 3:
We know nothing about the functions f and g other than their polymorphic types:

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

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.

[taken from exercise 4.6 from the course notes]

Lecture 5

Exercise 4:
Implement selection sort using ML.

The 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. If you are not familiar with the algorithm, look it up online.

Comment on the space and time complexity of your implementation.

[taken from exercise 5.2 from the course notes]

Exercise 5:
Implement bubble sort using ML.

The 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. If you are not familiar with the algorithm, look it up online.

Comment on the space and time complexity of your implementation.

[taken from exercise 5.4 from the course notes]

Exercise 6:

Research the Counting Sort algorithm. What is its time and space complexity? Explain why it does not contradict slide 502. What time/space complexity might counting sort have if implemented in ML using lists to store the counts?

Lecture 6

Exercise 7:

Write a datatype for representing the months of the year (look at the datatype for representing vehicles on slide 601 for help) . Write a function called `to_day` to convert from a month and day to the day number in the year[2] and a function called `to_date` to convert back from the day number in the year to a month and day.

For example, `to_day (Feb,2)` returns 33 and `to_day (Dec,31)` return 365 and `to_date 12` returns `(Jan,12)` and `to_date 284` return `(Oct,21)`.

If you are short of time, then assume all months have exactly 30 days and handle only the first 4 months.

[adapted from exercise 6.1 from the course notes]

Exercise 8:

The following is a datatype for representing lists.

```
datatype 'a mylist = Nil | Cons of 'a * 'a mylist
```

This definition was taken from page 68 (under slide 609) in the course notes.

(a) Write new `hd`, `tl` and `null` functions (called `myhd`, `mytl` and `mynull`) for this new representation of lists. Your `myhd` or `mytl` functions should handle empty list the same way as ML's `hd` and `tl` functions do.

(b) Write new append and reverse functions (called myappend and myrev) for this new representation of lists.

(c) Write the functions `to_mylist` and `to_list` which convert between this new representation of lists and the built-in representation of lists.

Exercise 9:

The ML datatype `Boolean`, defined below, is to be used to represent boolean expressions.

```
datatype Boolean =
    Var of string
  | Not of Boolean
  | And of Boolean * Boolean
  | Or of Boolean * Boolean
```

[2] if you are unsure what a day number is see http://www.epochconverter.com/epoch/daynumbers.php

The constructor `Var` is used to represent named boolean variables, and `Not`, `And` and `Or` are used to represent the corresponding boolean operators. Here are some example boolean expressions, written in ML using the `Boolean` datatype.

```
(* c = a.b *)
let c = And(Var("a"), Var("b"))

(* x = a.b + c.d *)
let x = Or( And(Var("a"), Var("b")), And(Var("c"), Var("d")))
```

(a) Define a function that will return a list of the names used in a given boolean expression.

(b) Define a function that will return a list of the **distinct** names used in a given boolean expression.

(c) A context is represented by a list of strings corresponding to the boolean variables that are set to true. All other variables are deemed to be set to false. Define a function called *eval* that will evaluate a given boolean expression in a given context.

```
(* what is a, when a=true *)
> eval (Var "a",["a"])
val it = true: bool

(* what is a, when a=false *)
> eval (Var "a",[])
val it = false: bool

(* what is a.b, when a=true and b=false *)
> eval (And(Var "a", Var "b"),["a"])
val it = false: bool

(* what is a+b, when a=true and b=false *)
> eval (Or(Var "a", Var "b"),["a"])
val it = true: bool
```

(d) Incorporate your two functions into a program that will determine whether a given boolean expression is true for all possible settings of its variables.

[taken from tripos question 1996 Paper 1 Question 2]

Exercise 10:

(Hard) Exercise 8 creates a datatype for boolean algebra and goes on to evaluate expressions in boolean algebra. We will now look at this in the context of arithmetic expressions instead.

(a) Give an ML datatype for representing arithmetic expression: an expression in our case is either a integer, the additions of two expressions, the multiplication of two expressions or the negation of an expression.

(b) Write a function called `eval` which transform an arithmetic expression to an integer e.g.

```
> eval (Add (Int 4, Int 6));
val it = 10: int
```

```
> eval (Multi (Add (Int 1, Int 2), Neg (Int 3)));
val it = ~9: int
```

(c) Extend your datatype and eval definition to allow for variables. A context is represented by a list of string integer tuples. An example of the results is:

```
> val context = [("x",7),("y",8)];
val context = [("x", 7), ("y", 8)]: (string * int) list

> eval (context, Add (Var "x", Var "y"));
val it = 15: int
```

Note that context is an example of an association list, which is a simple dictionary representation detailed on slide 701. If the expression contains any string which are not mapped to integers, your function should raise an exception.

[adapted from exercise 6.4 & 6.5 from the course notes]

4 Problem Sheet Three

This supervision is all about binary trees, functions as values, partial application and streams. Please use multiple argument functions wherever possible, instead of tuples.

This supervision will cover the material from lectures 7 to 9.

Lecture 7

Exercise 1:
Draw the binary search tree in 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?

[adapted from exercise 7.1 in the course notes]

Exercise 2:
Describe an algorithm for deleting an entry from a binary search tree. Comment on the suitability of your approach. Now code this algorithm, I have provided some familiar functions and example binary search trees in the `sup3-helper.sml` file.

[adapted from exercise 7.4 and 7.5 in the course notes]

Exercise 3:
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.

[adapted from exercise 7.8 in the course notes]

Lecture 8

Exercise 4:
Code the curried function `exf`, which takes as arguments the function `f` and the list `l`. The result must consist of those elements `x` of `l` such that `f(x)` is also a member of `l`. The elements of the result must be distinct from each other but may appear in any order. For example, if `f(x) = x + 1` and `l = [9, 3, 2, 2, 8]` then the result should be `[2, 8]` or `[8, 2]`.

[taken from 2000 P1 Q1]

Exercise 5:
The type `option`, declared below, can be viewed as a type of lists having at most one element. (It is typically used as an alternative to exceptions.) Write a new function that combines both map and filter, such that when the function given returns `None` then remove the element (like filter) and when the function given returns `Some x` then put `x` into the list (like map).

```
datatype 'a option = None | Some of 'a
val mapfilter = fn: ('a -> 'b option) -> 'a list -> 'b list
```

[taken from exercise 8.4 in the course notes]

Exercise 6:

This is past exam question, try (at least at first) to complete the question within 30 mins.

(a) The polymorphic curried function `delFirst` takes two arguments, a predicate (boolean-valued function) `p` and a list `xs`. It returns a list identical to `xs` except that the first element satisfying `p` is omitted; if no such element exists, then it raises an exception. Code this function in ML.

(b) Use the function `delFirst` to express the polymorphic function `delFirstElt`, where `delFirstElt x xs` returns a list identical to `xs` except that it omits the first occurrence of `x`.

(c) Carefully explain the polymorphic types of these two functions, paying particular attention to currying and equality

(d) A list `ys` is a permutation of another list `xs` if `ys` is obtained by rearranging the elements of `xs`. For example, `[2,1,2,1]` is a permutation of `[2,2,1,1]`. Code an ML function to determine whether one list is a permutation of another.

(e) A list `ys` is a generalised permutation of `xs` if `ys` is obtained by rearranging the elements of `xs`, where one element of `xs` is specially treated: it may appear any number of times (including zero) in `ys`. For example, `[1,2,1]` is a generalised permutation of `[1,2]` but `[1,2,2,1]` is not because two elements (1 and 2) appear the wrong number of times in it. Code an ML function to determine whether one list is a generalised permutation of another.

[taken from tripos question 2009 P1 Q1]

Lecture 9

Exercise 7:

For the first supervision we wrote a 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]`. Now write the equivalent for lazy lists.

[adapted from exercise 3.3 from the course notes]

Exercise 8:

A lazy binary tree is either empty or is a branch containing a label and two lazy binary trees, possibly to infinite depth

(a) Present an ML datatype to represent lazy binary trees

(b) Present an ML function that accepts a lazy binary tree and produces a lazy list that contains all of the trees labels.

[taken from exercise 9.4 in the course notes]

Exercise 9:

Write an analogue of map for sequences.

[taken from exercise 9.1 in the course notes]

5 Problem Sheet Four

All of these questions are real past exam questions, you can find the questions at `http://www.cl.cam.ac.uk/teaching/exams/pastpapers`. This supervision will cover the material from lectures 10 to 12 and review the materials from the rest of the course. Limit yourselves to around 30 mins per question to practice your timings for exams.

Exercise 1:
2002, Paper 1 Question 5 - for those not familiar with foldr, is is defined and used as follows:

```
1
2  (* given a function f, a list and accumulator, foldr recusively
3   applies the head of the list and accumulator to f *)
4  fun foldr f ([],    e) = e
5    | foldr f (x::xs, e) = f(x, foldr f (xs,e))
6
7  fun sum xs = foldr (fn (x,acc) => x+acc) (xs,0)
8
9  (* an example call trace for sum
10 assume f means (fn (x,acc) => x+acc)
11 sum [3,5] =>
12 foldr (f ([3,5],0)) =>
13 f (3, foldr f ([5],0))) =>
14 f (3,  (5, foldr f ([],0)))) =>
15 f (3,  (5, 0)) =>
16 f (3,5) =>
17 8
18 *)
```

Exercise 2:
2005, Paper 1 Question 1 - covering representing polynomials

Exercise 3:
2006, Paper 1 Question 5 - covering queues

Exercise 4:
2008, Paper 1 Question 6 - covering searching

Exercise 5:
2006, Paper 1 Question 6 - covering mutability and reviewing previous materials

6 Lent Revision Workshop

In preparation for the ML workshop, please work through the follow questions. They are designed to be fairly straight forward and refresh you knowledge after the Christmas break. Please try not to use your lecture notes and bring your solutions with you to the workshop.

Exercise 1:
Why might a programmer choose to use ML, instead of an imperative language such as Java or C++?

Exercise 2:
Write a function `xor` which takes two booleans and return true if either of them are true and false otherwise.

Exercise 3:
Write a function to compute the factorial of an integer, how can you adapt this for floating point values?

Exercise 4:
Solve the recurrence equations

$$T(0) = 1$$

and

$$T(n+1) = T(n) + n$$

Exercise 5:
Write a function to test if a given list is empty.

Exercise 6:
Write a function which given a list of integers, returns the sum of the integers in the list.

Exercise 7:
Write a function to reverse a list in $0(n)$.

Exercise 8:
Write a function which given a list, returns true if all the elements of the list are the same and false otherwise. What is the type of this function? Justify your answer.

Exercise 9:
Write a function called `zip`, which takes tuple of two lists and returns a list of tuples containing the elements of the lists, for example:

```
1 zip ( [1,2,3,4,5], [2,4,6,8,10] );;
2 > [ (1,2), (2,4), (3,6), (4,8), (5,10)]
```

Exercise 10:
Write a function to check if a list is sorted.

Exercise 11:
Write a function to merge two sorted lists.

Exercise 12:
Give a ML datatype for binary trees.

Exercise 13:
Give an ML datatype for ML's built-in lists.

Exercise 14:
Implement ML's `hd` function which returns the first element of a list or if the list is empty, throws an exception called `Empty`.

Exercise 15:
Write a function which given a integer binary tree, returns the sum of the integers in the binary tree.

Exercise 16:
An association list is a list of pairs which can be used for a simple implementation of a dictionary. Implement a dictionary as an association list with the following common dictionary operations: empty, lookup, update and delete.

Exercise 17:
Describe the structure of binary search trees. With a diagram, show how inserting the same set of keys in a different order can result in different shaped binary search tree.

Exercise 18:
Implement a dictionary using a binary search tree with the following dictionary options: empty, lookup and update.

Exercise 19:
Explain the difference between preorder, inorder and postorder tree traversal and write a function to do each (do not worry about efficiency).

Exercise 20:
What is a functional array? Write a function to lookup a value in a functional array.

Exercise 21:
Write a function called `map` which takes a function and a list and applies the function to each of the elements and returns the result.

Exercise 22:
Give the ML datatype for lazy lists.

Exercise 23:
Write a function which given a lazy list and a positive integer n, returns a normal ML list containing the first n elements of the lazy list.

Exercise 24:
Write a function called `interleave` which takes two lazy lists and joins them, interleave the elements from the two original lists.

Exercise 25:
Describe the difference between breadth first, depth first and iterative deepening tree traversal.

Exercise 26:
Implement breadth first and depth first tree traversal in ML, use lists for both and do not worry about efficient.

Exercise 27:
Describe how a pair of lists can be used to implement a queue in ML, include ML code for the common queue operations

Exercise 28:
Describe how we can represent univariate polynomials in ML using `(int * real) list`.

Exercise 29:
Comment, with examples, on the difference between an `int ref list` and an `int list ref`.

Exercise 30:
Write a version of map (from question 21) for arrays instead off lists.

7 Easter Revision Supervision

The purpose of this supervision is to help you to prepare for the upcoming exams. You have the best understanding of your current strengths and weaknesses so please step up and guide me as to how I can best support you. I am at your disposal so feel free to contact me over the next month should you have any difficulties or if you would like some feedback on practice tripos questions. For this supervision I have chosen four of the questions which students are particularly struggling with so please do not be scared off.

Foundations of Computer Science is in Section A of Paper 1[3]. Candidates attempt one of the two questions on the course. Each paper is 3 hours long, so this approximates to 30 minutes per question.

The best way to revise for the exam is to do timed practice questions. I would recommend predicting how many marks you have will get on a question before having a go at it and mark it. It's important to understand your own strengths and weaknesses to help you choose the best questions in the exam.

Exercise 1:

2008 Paper 1 Question 5: This questions covers laziness, in particular lazy lists and binary trees. Have a think about why would might use these data structures in reality. If you are struggling, start off by writing the functions for normal binary trees then transform then to lazy trees.

Exercise 2:

2012 Paper 1 Question 2: This question is on mutability with some bookwork on currying and partial application. I picked this question not only because its a personal favourite of mine but because students often struggle with these kind of reasoning questions.

Exercise 3:

2011 Paper 1 Question 1: This is a interesting question with few marks for bookwork.

Exercise 4:

2005 Paper 1 Question 1: This question is on polynomials. We discussed this in our fourth supervision but many students simply copied the materials on univariate polynomials from the notes.

[3]the full structure of the papers is online at `http://www.cl.cam.ac.uk/teaching/exams/exam-structure.pdf`