PART IA

Foundations of Computer Science

Supervision Exercises

Michaelmas Term 2026 · Supervisions 1–4

Michael Lee

Installing OCaml . . . 1

1 Local installation . . . 1
2 Configure an IDE . . . 1
3 Compile an OCaml program . . . 1
4 Link OCaml programs . . . 1
5 Dune: a build system for OCaml . . . 2
6 Utop: a read-eval-print loop . . . 3
7 Jupyter notebooks . . . 3
8 That's it! . . . 4

Supervision 1 . . . 5

1 Representations . . . 5
2 Abstraction barriers . . . 7
3 Floating point imprecision . . . 8
4 Functional programming basics . . . 9
5 Recursion and iteration . . . 10
6 Efficiency . . . 12
7 Lists . . . 12

Supervision 2 . . . 15

1 More on lists . . . 15
2 Sorting . . . 18
3 Trees . . . 19
4 Datatypes . . . 21

Supervision 3 . . . 23

1 Datatypes (continued) . . . 23
2 Dictionaries and functional arrays . . . 24
3 Functions as values . . . 26
4 Sequences and laziness . . . 30

Supervision 4 . . . 33

1 Sequences and laziness (continued) . . . 33
2 Queues and search strategies . . . 36
3 Elements of procedural programming . . . 38

HOW TO READ THESE SHEETS

●●●○○ Difficulty runs from one to five. Beside the estimated time is a description of the type of work. *Practical* is meant to build up a hard skill: in this course, mostly to train OCaml fluency; *conceptual* is hard but often has a short answer, relying on insight; *reflection* asks you to step back from the question and consider how it fits into a bigger picture; *open-ended* is a discussion question: meant to prompt questions and opinions. I do not expect correct answers, even if they exist.

★ A ★ in the margin means the question, or that part of the question, is optional.

OCaml A code panel is a link: click anywhere on it to open that code, and whatever it builds on, in an OCaml editor in your browser. Most are self-contained, minimally reproducible snippets. Some require you to fill in the blanks.

HINT Set in grey after each question.

SOLUTION

Only in the solutions build.

PICKED UP IN

PART IB Another Course · *what it calls the idea*

A block like this says where an idea is revisited: the part of the Tripos, the course, and the name that course gives it. In passing, the same pointer is written inline as (IB ANOTHER COURSE).

COURSE OBJECTIVES

In my opinion, the course has three aims, in descending order of importance:

- **Thinking mathematically**. At the end of the course, you should start seeing programming and maths as exercising the same skills. Maths isn't just useful when proving things about programs, although it's important there, too. I hope you start to see programs in the language of maths, and data structures as fundamentally mathematical (inductive) definitions. Thinking mathematically about programs helps you write clear and correct programs.
- **Becoming fluent in OCaml**. You should be able to write correct, or near-correct, programs in OCaml without much assistance — correct in both senses: the program compiles, and it does what you meant. OCaml's roots are mathematical, so thinking mathematically will help you write better OCaml. What it won't help you do is gain fluency with its syntax. The compiler will: it gives you immediate, rich feedback about your programs, and tests them in ways your brain cannot. Somewhat unintuitively, then, program with the compiler rather than on pen and paper, as much as you can.
- **Reasoning about cost**. One of the reasons many programmers like OCaml is that, unlike other languages, it is often very easy to reason about the cost of a program. Beyond seeing why your program is clear and correct, I hope you will also be able to reason about its cost. As a software engineer, most programs need to be clear, few need to be fast. But that should be a deliberate choice.

PHILOSOPHY

I have two competing aims as a supervisor. First, I want to equip you with the skills you need to perform well in the exams, for future courses / projects, and for prospective careers in research and/or industry. Second, I want to set a reasonable amount of work, so as not to overload you.

These sheets contain many things not taught explicitly in the course, or in the examinable material. I include them because I believe they are essential for your future education, and because the Tripos — not just Foundations of Computer Science — rewards students who study the material deeply, and push themselves beyond what they are taught in lecture.

To keep that from becoming a burden, I have tried to engineer the questions carefully. The sheets may look long, with many questions, but that is because I am trying to guide you carefully through a difficult concept. Many of my questions can be answered in just one or two lines.

I have also tried to be upfront about the workload: difficulty ratings, and estimated time per question, section and sheet.

If you ever feel like the work is unreasonably difficult or that there is simply too much of it, please email me.

A LITTLE ADVICE

When I was a student, I found the following pieces of advice helpful:

- **Prioritise the sheet**. I have an hour per supervision with you. Each sheet is about 6 hours long. In school, most of my time was spent in class, and less doing the homework: it's different at university.
- **Start early**. Pace yourself, and be proactive. Since we don't have much time together, it's important to make every minute count — that means starting the work early, so you've had time to think about it before we talk.
- **Think carefully, and have fun**. These sheets are not graded. I care that you think carefully about your answers. A thoughtful question is *better* than a correct answer with no sign of thinking (or worse, obvious signs of AI usage). I care that you have fun — I hope to set questions that show computer science is so much more than what you see on Leetcode or StackOverflow.
- **Help each other**. Learning is a communal activity. Oftentimes, your friends will have insights and perspectives that you won't — and vice versa — sharing knowledge is the best way of gaining knowledge.
- **Ask questions**. It's fine to ask me about questions on the sheet that you don't understand before the supervision. I might not respond with a full answer, but I might give you a hint. Ask me questions during the supervision. I'd encourage you not to think of me as your teacher, but as your employee. I'll prepare content for the supervision, but knowing how you're doing with the content helps me tailor my teaching to you as an individual. I can't answer questions that you don't ask.

ACKNOWLEDGEMENTS

The questions on these sheets build on those by Dr. Dima Szamozvancev, as well as the course notes. Any errors in the setting or the solutions are mine; corrections are welcome.

FOUNDATIONS OF COMPUTER SCIENCE

Installing OCaml

Michaelmas Term 2026 · Michael Lee

1 Local installation

While the course provides a helpful online sandbox, this is an opportune moment to install OCaml on your computer. No exercises will require a local installation of OCaml, but some will be easier with a local installation. Please try and complete the full installation, including the recommended libraries.

2 Configure an IDE

Following installation, you should configure an IDE of your choosing to work with OCaml.

3 Compile an OCaml program

To test that everything is working, I have provided a `hello.ml` file, which contains the following code:

```OCaml
open Printf;;

let hello n = printf "hello %s %d\n" "world" n;;
```

On the command line, execute the following command:

```Shell
ocamlc hello.ml
```

The `ocamlc` command compiles the OCaml code, creating an executable file `a.out`.

4 Link OCaml programs

In the same folder, there should be a file called `main.ml`, with the following code:

```OCaml
open Hello;;

hello 123;;
```

The line `open Hello` is equivalent to the `import` statement that you might find in Python. In this case, it imports the `Hello` module defined in `hello.ml`

Now run the following command on your terminal:

```Shell
ocamlc main.ml
```

You should get an error:

```Shell
File "main.ml", line 1:
Error: No implementation provided for the following modules:
         Hello referenced from Main (main.cmo)
```

`main.ml` depends on `hello.ml`, and we need to explicitly give `ocamlc` the `hello.ml` file. This process is known as linking. The following command should succeed:

```Shell
ocamlc hello.ml main.ml
```

You can now execute `main.ml` by typing

```Shell
./a.out
```

which should print `hello world 123`

The order here is important! `main.ml` depends on `hello.ml`, so it must come **after** `hello.ml`. That is, the following command:

```Shell
ocamlc main.ml hello.ml
```

will raise the following error:

```Shell
File "hello.ml", line 1:
Error: Wrong link order: Main (main.cmo) depends on Hello (hello.cmo)
```

---

5 Dune: a build system for OCaml

It can be difficult to organise the various files (modules) into the right order for compilation and linking. A **build system** is a system that helps you automate the process of compilation and linking. Many languages have build systems (for example, Maven for Java). Dune is a build system for OCaml. You might want to familiarise yourself with Dune.

You can initialise a new project by executing

```Shell
dune init project test
```

If your terminal complains that it doesn't know what `dune` means, try running

```shell
eval $(opam env)
```

then run `dune init project test`.

We can now repeat the example above! Create a new `hello.ml` file under `test/lib`, and copy and paste the code in `hello.ml` into it. In `bin/main.ml`, just write

```ocaml
open Test;; (*or whatever you named your project*)
open Hello;;

hello 123;;
```

Then you can do

```shell
dune build
dune exec main
```

and the program will be compiled and run for you!

---

6 Utop: a read-eval-print loop

If you would like a more interactive programming experience, you can use `utop` (universal top-level). A good introduction to `utop` can be found here.

To test `utop` out, you can execute the following command

```shell
dune utop
```

Inside utop, you can load a file as follows

```shell
#use "bin/main.ml";;
```

Remember to end every line with `;;`

To quit, just type `#quit;;`

---

7 Jupyter notebooks

You might also find it helpful to set up Jupyter notebook such that you can create OCaml notebooks.

Here's how you can do that.

---

8 That's it!

If anything is unclear, or you got stuck along the way, please reach out! We can schedule a call to get you set up.

FOUNDATIONS OF COMPUTER SCIENCE

Supervision 1

Michaelmas Term 2026 · Michael Lee

WORK DUE	A single PDF, 24 hours before the supervision
EXPECTED TIME	6 hours 5 min

NAMES IN SCOPE

In this sheet, functions come from one of three places:

1. Defined in-line
2. Defined in the lecture notes (e.g. `member`, `zip`, `unzip`)
3. Defined in the OCaml standard library (e.g. `List.map`)

If your editor complains that a function has been used but has not been defined, please copy in the definition from the notes.

NOTE

Sections 1 and 2 assume you have successfully installed OCaml and Dune, by following the instructions of the installation guide. **If you are unable to install OCaml or Dune, please email me as soon as you can.**

1 Representations

PREREQUISITES Lecture 1 · TIME 40 min

Good representations make it easy to see why a solution is correct, and easy to maintain when the problem changes. Bad representations, in contrast, tend to result in brittle, bug-prone, and hard-to-maintain solutions.

QUESTION 1

PRACTICAL · 40 MIN ●●○○○ BASIC

A hands-on introduction to the OCaml module system.

One solution to the year 2000 bug mentioned in Lecture 1 involves storing years as two digits, but interpreting them so that `50` means 1950, `0` means 2000 and `49` means 2049.

(a) Have a look at the `representations/lib` directory, and in particular at the file `two_digit.ml` (see here for information about code organisation). Execute these commands:

```shell
cd representations
dune build
```

You should see an error:

```
File "lib/two_digit.ml", line 1:
Error: The implementation lib/two_digit.ml does not match the interface
       lib/.representations.objs/byte/representations__Two_digit.cmi:
       The type year is required but not provided
       The value make is required but not provided
       The value get is required but not provided
       The value is_gteq is required but not provided
       The value add is required but not provided
```
Shell

Dune is complaining that some types and values are not provided in `two_digit.ml`. We will fix that by answering the following questions.

(b) `The type year is required but not provided`

Choose a representation for the `year` type that allows years to be stored as two digits, by writing `type year = ?`.

(c) `The value make is required but not provided`

Write a `make` function. `make` should take an integer and return a `year`, raising an exception if the integer is less than 1950 or greater than 2049. You can declare an exception with `exception Year_out_of_bounds_exception` and raise it with `raise Year_out_of_bounds_exception`.

(d) `The value get is required but not provided`

Write a `get` function. `get` should take a `year` and return a full four digit integer `get (make 1987) = 1987`.

(e) `The value is_gteq is required but not provided`

Write an OCaml function `is_gteq` to compare two years, just as `>=` compares integers.

(f) `The value add is required but not provided`

Write an OCaml function `add : year -> int -> year` to add or subtract a given number of years from another year. Raise an exception if the result falls outside 1950–2049.

(g) To test your answers, you should write some programs (do not submit these).

Write your answers in `representations/bin/dates.ml`. You can execute `dates.ml` by running `dune exec dates` in the terminal. Alternatively, you can navigate to the `representations/bin` directory, and execute `dune utop` in the terminal.

THE REPL

`dune utop` opens a REPL (read-eval-print loop), an interactive environment for executing OCaml. Once you've opened the REPL, run `open Representations;;` and `open Two_digit;;`. This imports your code. Remember to end every statement with `;;`. Use `#quit;;` to quit the REPL once you're done.

(h) Comment on the merits and demerits of the two digit representation. In what situations might the two digit representation be useful?

2 Abstraction barriers

PREREQUISITES Lecture 1 · TIME 40 min

Concepts change much more slowly than implementation. An abstraction barrier lets you define concepts. Implementers of the abstraction barrier implement these concepts as implementations. Consumers see and use only the concept, not the implementation. This divide allows the implementation to change internally, without impacting consumers that rely only on the concept.

QUESTION 2

PRACTICAL · 10 MIN ●○○○○ BASIC

A hands-on introduction to OCaml signatures.

In OCaml, abstraction barriers are enforced by *signatures*. This question will walk you through the creation of a signature for your `two_digit.ml` file.

(a) There should be a `representations/lib/two_digit.mli` file, containing:

```
type year
val make : int -> year
val get : year -> int
val is_gteq : year -> year -> bool
val add : year -> int -> year
```

This is known as a *signature*. A signature for an OCaml file tells other files that `two_digit.ml` exports a type `year`, and four functions, `make`, `get`, `is_gteq`, and `add`. This signature is the reason why Dune was complaining earlier, when these functions had been promised, but not actually implemented.

(b) Extend your earlier tests with a test that tries to manipulate the underlying representation of `years`. For example, if `years` are represented as integers, you could try to manipulate the integer by writing

OCaml
```
(* ... *)
open Two_digit;;
(make 1950) + 1;;
```

Does OCaml let you write this expression? What is being abstracted over?

(c) Replace `type year` in the signature (`two_digit.mli`, not `two_digit.ml`) with

OCaml
```
(*or whatever type you chose to use for the representation*)
type year = int
```

Now repeat part (b) and report your results. What is the difference between `type year` and `type year = int`?

(d) Undo your change to the signature, so that it reads `type year` once more, and check that your tests still compile.

QUESTION 3

PRACTICAL · 20 MIN ●○○○○ BASIC

(a) Repeat question 1, but with a four digit representation of years. Your code should raise an error for years before 1950 and after 2049.

Create `four_digit.ml` and `four_digit.mli` in `representations/lib`, alongside `two_digit.ml`. `four_digit.mli` should be exactly the same as `two_digit.mli`. `four_digit.ml` should implement the same type and functions using four digits rather than two.

(b) Modify your test code to use `Four_digit` rather than `Two_digit`. That is, replace `open Two_digit` with `open Four_digit`. Run the tests, as before. Do you expect to have to make any changes? Why, or why not?

QUESTION 4

REFLECTION · 10 MIN ●●○○○ BASIC

Using your experience from questions 2 and 3, what is the main idea behind *abstraction barriers*? Why are they useful?

3 Floating point imprecision

PREREQUISITES Lecture 1 · TIME 10 min

*A good starting point for representations is: "the ones covered in A-level maths". Unfortunately, computers cannot encode all representations, only finite representations. In this section we look at $\mathbb{R}$, which computers **approximate** using floating point numbers.*

QUESTION 5

PRACTICAL · 5 MIN ●○○○○ BASIC

Seeing a decimal that binary cannot hold, and watching the error accumulate.

Because computers represent numbers as binary strings, simple decimals such as 0.1 often cannot be represented exactly. Write a function that computes

$$\underbrace{x + x + \cdots + x}_{n}$$

where x has type `float`. It is essential to use repeated addition rather than multiplication. See what happens when you call it with $n = 1\,000\,000$ and $x = 0.1$.

QUESTION 6

PRACTICAL · 5 MIN ●○○○○ BASIC

Another example of the inaccuracy of floating-point arithmetic takes the golden ratio $\phi = 1.618\ldots$ as its starting point:

$$\gamma_0 = \frac{1+\sqrt{5}}{2} \qquad \text{and} \qquad \gamma_{n+1} = \frac{1}{\gamma_n - 1}$$

In theory, it is easy to prove that $\gamma_n = \cdots = \gamma_1 = \gamma_0$ for all $n > 0$. Write this computation in OCaml and report the value of γ_{50}.

HINT In OCaml, $\sqrt{5}$ is written `sqrt 5.0`.

4 Functional programming basics

PREREQUISITES Lecture 1 · TIME 30 min

An introduction to functional thinking.

QUESTION 7

CONCEPTUAL · 5 MIN ●○○○○ BASIC

Why is it silly to write an expression of the form `if b then true else false`? What about `if b then false else true`? How about `if b then 5 else 5`?

QUESTION 8

CONCEPTUAL · 10 MIN ●●○○○ BASIC

Briefly discuss the meaning of the terms (pure) *expression*, *value*, *command* and *effect*, using the following examples.

- `true`
- `57 + 9`
- `print_string "Hello world!"`
- `print_float (8.32 *. 3.3)`

QUESTION 9

CONCEPTUAL · 10 MIN ●●○○○ BASIC

Which of these is a valid OCaml expression and why? Assume that variable `x` has previously been declared, e.g. with `let x = 1`.

- `if x < 6 then x + 3 else x + 8`
- `if x < 6 then x + 3`
- `if x < 6 then print_string "hello"`

- `if x < 6 then x + 3 else "A"`
- `x + (if x < 6 then 3 else 8)`

HINT Use the OCaml compiler to check your answers

QUESTION 10

CONCEPTUAL · 5 MIN ●○○○○ **BASIC**

Comment on the similarities and differences between:

(a) the `if` conditional in OCaml;

(b) the `if` conditional in Python (to be precise, the if statement, not the inline `b if c else d`);

(c) the `if` conditional in Java, if you have met it;

(d) the `a ? b : c` ternary operator in Java, if you have met it.

---

5 Recursion and iteration

PREREQUISITES Lecture 2 · **TIME** 1 hour 45 min

You may have heard that recursion is elegant and, in contrast, iteration is efficient. These questions are about seeing why that is only half right.

QUESTION 11

PRACTICAL · 5 MIN ●○○○○ **BASIC**

The triangular numbers are the sequence $0, 1, 3, 6, 10, 15, \ldots$. Implement `triangle` in OCaml, so that `triangle n` calculates the n^{th} triangular number, that is:

$$\begin{aligned} \text{triangle}\,0 &= 0 \\ \text{triangle}\,1 &= 1 \\ \text{triangle}\,2 &= 3 \\ \text{triangle}\,3 &= 6 \\ &\ldots \end{aligned}$$

QUESTION 12

PRACTICAL · 10 MIN ●●○○○ **BASIC**

Code an iterative (tail-recursive) version of the efficient `power` function from Section 1.6.

QUESTION 13

PRACTICAL · 10 MIN ●○○○○ **BASIC**

Write an iterative version of your answer to the previous question in Python, using a loop. Try to make your answer look similar to your answer to question 12.

QUESTION 14

CONCEPTUAL · 20 MIN ●●●●○ CORE

Using your answer to the previous question, devise a procedure for converting a tail-recursive algorithm into an iterative one. You may assume the base cases and recursive cases of the tail-recursive algorithm can be identified. Your procedure should explain how to treat:

(a) The arguments to the function (in the previous question, `x`, `n`, and `acc`)

(b) The base cases (when `n == 0`. In general, there might be more than one base case)

(c) The recursive cases (when `n > 0`. In general, there might be more than one recursive case).

QUESTION 15

CONCEPTUAL · 30 MIN ●●●○○ CORE

(a) Write down a definition for tail recursive functions. Your definition should begin with "a tail recursive function is"

(b) Give an example of a non-tail recursive function, and explain why your definition precludes it.

(c) Are the following functions tail recursive according to your definition? Explain why, or why not.

- `let id x = x`
- `let rec f () = f ()`
- `let rec f x acc = if x = 0 then 0 else (f (x-1) (acc + x); 0)`
- `let rec f x acc = if x = 0 then 0 else id (f (x-1) (acc + x))`
- `let rec f x acc = if x = 0 then 0 else ((); f (x-1) (acc + x))`

(d) If you have previous understanding of how functions are evaluated in a computer (stack frames), can you explain why tail-recursive functions are often more space-efficient than recursive ones?

QUESTION 16

OPEN-ENDED · 30 MIN ●●●●● ADVANCED

Are there recursive functions that cannot be written in a tail recursive fashion? Justify your answer by either demonstrating a function that cannot be written in tail recursive style, or explain why all recursive functions can be written in a tail recursive fashion.

HINT This one is genuinely hard, and I will be very surprised if anyone correctly answers it. For open-ended questions like these, I suggest a strategy: read it, think about it for at least the time indicated above, do something else (finish the sheet, go on a walk, meet friends) while it marinates in the back of your mind, and then come back to it several hours / days later.

6 Efficiency

PREREQUISITES Lecture 2 · TIME 55 min

This section aims to familiarise you with big-O notation and recurrence relations.

QUESTION 17

CONCEPTUAL · 30 MIN ●●●●○ CORE

Consider an arbitrary non-negative integer k. Set n to be 2^k. Use a *recurrence relation* to find a closed form for the recurrence given by $T(1) = 1$ and $T(n) = 2T(n/2) + 1$, and hence a bound on its growth. You should be able to find a tighter bound than $O(n \log n)$. Prove that your closed form is correct for all k using mathematical induction.

QUESTION 18

CONCEPTUAL · 25 MIN ●●●●○ CORE

Let $a_1, \ldots, a_k$ be constants with $a_i > 0$, and let $g_1, \ldots, g_k$ be functions such that $g_i(n) \geq 0$ for $i = 1, \ldots, k$ and all sufficiently large n. Show that if $f(n) = O(a_1 g_1(n) + \cdots + a_k g_k(n))$ then $f(n) = O(g_1(n) + \cdots + g_k(n))$.

---

7 Lists

PREREQUISITES Lecture 3 · TIME 1 hour 25 min

Lists are one of the fundamental data structures of functional programming. In this section, you should develop a basic grasp for programming with lists.

QUESTION 19

CONCEPTUAL · 15 MIN ●●●○○ CORE

There are two competing definitions for "a list of type T"

Def 1. A list of type T is some sequence of elements $a, b, c, \ldots$, all of type T.

Def 2. A list of type T is one of the following:

- The empty list, []
- An element of type T, x, and a list of type T, xs, joined together $x :: xs$.

Assume we are given a list of type T.

(a) According to Def 1, how should we process the elements of the list?

(b) According to Def 2, how should we process the elements of the list?

(c) Identify which of the definitions is imperative, appealing to a physical description of how elements might be laid out.

(d) Identify which of the definitions is functional, appealing to a mathematical description for constructing lists.

QUESTION 20

CONCEPTUAL · 10 MIN ●○○○○ BASIC

Explain why seeing the following expressions in OCaml code should be a cause for concern:

- `1 :: [2, 3, 4]`
- `"hello " @ "world"`
- `xs @ [x]`
- `[x] @ xs`
- `List.hd xs + List.length (List.tl xs)`

QUESTION 21

PRACTICAL · 10 MIN ●●○○○ BASIC

Write a recursive and an iterative function to compute the sum of a list's elements. Compare their relative efficiency.

HINT For FoCS, when you see "efficiency", it is a directive to use big-O notation.

QUESTION 22

PRACTICAL · 10 MIN ●●○○○ BASIC

Write a function to return the last element of a non-empty list. How efficiently can this be done? See if you can come up with two different solutions.

QUESTION 23

PRACTICAL · 10 MIN ●●○○○ BASIC

Write a function `odds` that keeps the odd-indexed elements of a list, and discards the rest. For example, given `[a; b; c; d]` (whose indices are 0, 1, 2 and 3) it should return `[b; d]`.

HINT Pattern-matching is a very flexible concept.

QUESTION 24

PRACTICAL · 15 MIN ●●○○○ BASIC

Code a function `tails` to return the list of the tails of its argument. For example, given the input list `[1; 2; 3]` it should return `[[1; 2; 3]; [2; 3]; [3]; []]`.

QUESTION 25

PRACTICAL · 15 MIN

●●○○○ BASIC

(a) Write a function `sum` that returns the sum of the elements of an integer list

(b) Write a function `mul` that returns the product of the elements of an integer list

(c) What can you say about your answers to parts a and b? Is there a way of sharing the majority of their implementations? Why might sharing be good?

FOUNDATIONS OF COMPUTER SCIENCE

Supervision 2

Michaelmas Term 2026 · Michael Lee

WORK DUE	A single PDF, 24 hours before the supervision
EXPECTED TIME	6 hours 10 min (+ 15 min for the optional questions)

1 More on lists

PREREQUISITES Lecture 4 · TIME 1 hour 45 min (+ 15 min)

This section aims to help you get really familiar with lists.

QUESTION 1

CONCEPTUAL · 15 MIN ●●○○○ BASIC

This question, and the next, use a slightly different version of `zip` than in the notes. Specifically, `zip` in the notes is curried

```
val zip: 'a list -> 'b list -> ('a * 'b) list
```

Whereas the `zip` in this question is uncurried

```
val zip: 'a list * 'b list -> ('a * 'b) list
```

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`

QUESTION 2

CONCEPTUAL · 10 MIN ●●○○○ BASIC

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`?

QUESTION 3

PRACTICAL · 25 MIN ●●●○○ CORE

(a) Use the `member` function of Section 4.2 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) Package your answers above into a module with the following signature. Choose the right implementation of `t`

```OCaml
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
```

Submit only your choice of `t`, not the full implementation.

(d) OCaml's implementation of Set requires that two elements of a set can be *compared*: that is, for every two elements `x` and `y`, we can establish if $x \leq y$. Why might this stronger requirement allow for more efficient implementations of sets?

HINT For part (d), you might need to wait for lecture 6.

★ QUESTION 4

CONCEPTUAL · 15 MIN ●●●○○ CORE

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 `Int_set` using

```OCaml
module Int_set = Set.Make(struct
  type t = int
  let compare a b = (* custom comparator here *)
end)
```

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

QUESTION 5

PRACTICAL · 15 MIN ●●○○○ BASIC

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?

QUESTION 6

CONCEPTUAL · 10 MIN ●○○○○ BASIC

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

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

QUESTION 7

PRACTICAL · 10 MIN ●●○○○ BASIC

Write a function `plusl` that computes the point-wise addition on two lists. For example,

OCaml
```
plusl [1;2;3] [4;5;6] (* [5;7;9] *)
```

QUESTION 8

CONCEPTUAL · 20 MIN ●●●○○ CORE

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 ; 0]`

(e) The `amt` is set to `-1`

---

2 Sorting

PREREQUISITES Lecture 5 · **TIME** 1 hour 15 min

Algorithms should cover sorting much more thoroughly, so you should treat this as a gentle introduction to algorithmic thinking, rather than a comprehensive set of questions.

QUESTION 9

CONCEPTUAL · 15 MIN ●●○○○ **BASIC**

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?

QUESTION 10

CONCEPTUAL · 15 MIN ●●○○○ **BASIC**

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

---

QUESTION 11

PRACTICAL · 15 MIN ●●●○○ **CORE**

Implement bubble sort in OCaml.

QUESTION 12

PRACTICAL · 20 MIN ●●●●○ **CORE**

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.

QUESTION 13

REFLECTION · 10 MIN ●●○○○ BASIC

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 · TIME 1 hour 50 min

Trees are an important functional data structure, alongside lists. This section aims to both teach you about trees, and uncover their underlying mathematical structure as inductively-defined data.

QUESTION 14

CONCEPTUAL · 10 MIN ●●○○○ BASIC

Examine the following function declaration. What does `ptree 1 n` accomplish?

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

QUESTION 15

PRACTICAL · 15 MIN ●●○○○ BASIC

(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?

QUESTION 16

CONCEPTUAL · 20 MIN ●●●○○ CORE

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

$$\frac{}{x \in X} \qquad \frac{x_1 \in X \quad x_2 \in X \quad \cdots \quad x_n \in X}{y \in X}$$

The first states that x must be in X. The second 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:

$$\frac{}{0 \in \mathbb{N}} \qquad \frac{n \in \mathbb{N}}{s(n) \in \mathbb{N}}$$

The first states that 0 is a natural number. The second 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.

(a) 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.

(b) Let Tree T be the set of trees of type T (this defines a *type-indexed* set). Generalise your answer to binary trees of type T.

QUESTION 17

CONCEPTUAL · 20 MIN ●●●○○ CORE

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.

QUESTION 18

CONCEPTUAL · 10 MIN ●●○○○ BASIC

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

QUESTION 19

CONCEPTUAL · 35 MIN ●●●●○ CORE

Prove the inequality involving the depth and size of a binary tree `t` from Section 6.10.

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

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 what you are assuming and what you are proving!

---

4 Datatypes

PREREQUISITES Lecture 6 · TIME 1 hour 20 min

An introduction to OCaml algebraic datatypes, and their uses. Revisits lists, now cast in the vocabulary of datatypes.

QUESTION 20

CONCEPTUAL · 15 MIN ●●○○○ BASIC

You have seen how binary trees can be defined as a datatype

```OCaml
type 'a tree = Lf | Br of 'a * 'a tree * '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

(d) Define lists of type T inductively, as you did for trees.

(e) Compare your answers in parts a and d.

QUESTION 21

CONCEPTUAL · 30 MIN ●●○○○ BASIC

(a) Translate the list $[0;1;2;3]$ into your inductive definition of lists (Question 20)

HINT It should look like Cons(...)

(b) Consider the alternative inductive definition of lists, which I call snoc lists (For clarity, for this question, I call normal lists cons lists):

$$\frac{}{\text{Nil} \in \text{SnocList}\, T} \qquad \frac{xs \in \text{SnocList}\, T \quad x \in T}{\text{Snoc}(xs, x) \in \text{SnocList}\, T}$$

(i) Translate the list $[0;1;2;3]$ into your inductive definition of SnocLists.

(ii) Define a type alias `'a clist` for `'a my_list`

(iii) Define an OCaml datatype `'a slist` for snoc lists

QUESTION 22

PRACTICAL · 20 MIN ●●●○○ CORE

(a) Define `mirror_cs: 'a clist -> 'a slist` such that

```OCaml
mirror_cs (Cons (0, Cons (1, Cons(2, Cons(3, Nil)))))
  = Snoc (Snoc (Snoc (Snoc (SNil, 3), 2), 1), 0)
```

(b) Define `mirror_sc : 'a slist -> 'a clist` such that

```ocaml
mirror_sc (Snoc (Snoc (Snoc (Snoc (SNil, 0), 1), 2), 3))
  = (Cons (3, Cons (2, Cons(1, Cons(0, Nil)))))
```

(c) Define `reassoc_cs: 'a clist -> 'a slist` such that

```ocaml
reassoc_cs (Cons (0, Cons (1, Cons(2, Cons(3, Nil)))))
  = Snoc (Snoc (Snoc (Snoc (SNil, 0), 1), 2), 3)
```

(d) Define `reassoc_sc: 'a slist -> 'a clist` such that

```ocaml
reassoc_sc (Snoc (Snoc (Snoc (Snoc (SNil, 0), 1), 2), 3))
  = Cons (0, Cons (1, Cons(2, Cons(3, Nil))))
```

HINT For the `reassoc` functions, it might be easier to define a helper function that uses an accumulator

---

QUESTION 23

CONCEPTUAL · 15 MIN ●●●●○ **CORE**

For each of the following functions, give the type of the function, and describe it.

(a) `fun xs -> mirror_sc (mirror_cs xs)`

(b) `fun xs -> mirror_cs (mirror_sc xs)`

(c) `fun xs -> reassoc_sc (reassoc_cs xs)`

(d) `fun xs -> reassoc_sc (mirror_cs xs)`

(e) `fun xs -> mirror_sc (reassoc_cs xs)`

HINT Acceptable descriptions look like “it is the identity function” and “it reverses the argument”.

FOUNDATIONS OF COMPUTER SCIENCE

Supervision 3

Michaelmas Term 2026 · Michael Lee

WORK DUE	A single PDF, 24 hours before the supervision
EXPECTED TIME	6 hours 20 min (+ 55 min for the optional questions)

1 Datatypes (continued)

PREREQUISITES Lecture 6 · TIME 1 hour

An introduction to OCaml algebraic datatypes, and their uses. Revisits lists, now cast in the vocabulary of datatypes.

QUESTION 1

CONCEPTUAL · 10 MIN ●●○○○ BASIC

With reference to

```OCaml
type ty = A of int * bool | B of string | C
```

explain why OCaml's datatypes are called algebraic datatypes.

HINT You might find this resource, and the phrase "sums of products" useful.

QUESTION 2

PRACTICAL · 10 MIN ●●○○○ BASIC

Give the declaration of an OCaml datatype for arithmetic expressions that have the following possible shapes: integers, booleans, or expressions of the form

$$-E \quad \text{or} \quad E \times E \quad \text{or} \quad E \leq E \quad \text{or} \quad \text{if } E \text{ then } E \text{ else } E$$

and write the expression

$$\text{if } 3 \leq 1 \text{ then } 4 \times 5 \text{ else } -7$$

QUESTION 3

PRACTICAL · 15 MIN ●●○○○ BASIC

Continuing the previous exercise, write a function `eval : expr -> expr` that evaluates an expression to return a value. As a sanity check, the example in the previous exercise should evaluate to `Int (-7)`. Some expressions, like `-true`, are nonsense: they have no meaning. Raise an `Ill_typed` exception on encountering these.

HINT Nested match blocks have to be wrapped in either parentheses () or `begin ... end`

QUESTION 4

PRACTICAL · 5 MIN ●○○○○ BASIC

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`.

QUESTION 5

CONCEPTUAL · 20 MIN ●●●●○ CORE

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`.

(a) 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`?

(b) In OCaml, the `unboxed` annotation tells the compiler to use Option 1 instead of Option 2. Why might it make sense to add such annotations to the language?

2 Dictionaries and functional arrays

PREREQUISITES Lecture 7 · TIME 2 hours 10 min

Lists and trees are the tools of the functional programmer. Arrays and dictionaries are the tools of the imperative one. This section shows how one can encode the second pair using the first.

QUESTION 6

CONCEPTUAL · 20 MIN ●●○○○ BASIC

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?

QUESTION 7

PRACTICAL · 15 MIN ●●○○○ BASIC

Write 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 of Section 7.4 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?

QUESTION 8

PRACTICAL · 30 MIN ●●●●○ CORE

Write an algorithm for deleting an entry from a binary search tree. Comment on the suitability of your approach.

HINT There are two reasonable methods — one is simple, the other is efficient but a bit more tricky.

QUESTION 9

PRACTICAL · 30 MIN ●●●●● ADVANCED

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.

QUESTION 10

CONCEPTUAL · 5 MIN ●●○○○ BASIC

Show that the functions `preorder`, `inorder` and `postorder` of Section 7.5 all require $O(n^2)$ time in the worst case, where n is the size of the tree.

QUESTION 11

CONCEPTUAL · 30 MIN ●●●●● ADVANCED

Difference lists represent lists as a higher-order function. The list `[0;1;2;3]` is written

```OCaml
fun xs -> 0::1::2::3::xs
```

(a) Write down the type of difference lists

```
type 'a dlist = (* fill this in*)
```

(b) Write a function that turns a difference list into a regular list

(c) Write a function that constructs the empty difference list `[]`

(d) Write a function that constructs the singleton difference list `[x]`

(e) Write a function that appends two difference lists. What is the time complexity of an append?

(f) Rewrite `preorder` to return a difference list, which we then convert back to a list.

(g) Analyse the time complexity of the rewritten `preorder`.

---

3 Functions as values

PREREQUISITES Lecture 8 · TIME 1 hour 50 min (+ 55 min)

A first taste of the power of functional programming, and the strong constraints imposed by polymorphism.

QUESTION 12

PRACTICAL · 10 MIN ●●●○○ CORE

Assume `t` is some known, but fixed, type. You are given a function `compare_l: t -> t -> int` such that the following equations hold

$$\begin{aligned} \texttt{compare_l x y} &= 0 \quad \text{when } x = y \\ \texttt{compare_l x y} &< 0 \quad \text{when } x < y \\ \texttt{compare_l x y} &> 0 \quad \text{when } x > y \end{aligned}$$

Write a function `compare_g` that satisfies the following equations

$$\begin{aligned} \texttt{compare_g x y} &= 0 \quad \text{when } x = y \\ \texttt{compare_g x y} &> 0 \quad \text{when } x < y \\ \texttt{compare_g x y} &< 0 \quad \text{when } x > y \end{aligned}$$

QUESTION 13

CONCEPTUAL · 10 MIN (+ 20 MIN) ●●●●● ADVANCED

Consider the following polymorphic functions.

OCaml

```
let sw f x y = f y x
let co g f x = g (f x)
let cr f a b = f (a,b)
```

```
val id  : 'a -> 'a = <fun>
val ap  : ('a -> 'b) -> 'a -> 'b = <fun>
val ucr : ('a -> 'b -> 'c) -> 'a * 'b -> 'c = <fun>
```

(a) Infer the types of `sw`, `co` and `cr` (without asking OCaml)

(b) Using only their types, give the definitions of `id`, `ap` and `ucr`.

(c) Give short descriptions for each of these functions.

(d) Reflect on your solutions to the previous two sheets: could any of your answers be improved by these higher order combinators? Do not submit your answers.

★ (e) Write the standard library composition operators in terms of the combinators above.

HINT The functions introduced in this exercise are all *higher-order function combinators*: functions that take functions as arguments and return functions as results. Further, they are all polymorphic, so their implementations are uniquely determined by the type.

QUESTION 14

CONCEPTUAL · 10 MIN ●●●●● **ADVANCED**

Polymorphic functions are highly constrained. You can often reconstruct a polymorphic function that you know nothing else about.

In this question, you can assume that all code is pure. That is, `f` and `k` do not have side effects. Assume a function `f` whose type is polymorphic in `'r`:

```
val f: (int -> 'r) -> 'r
```

(a) Give two different functions with this type `(int -> 'r) -> 'r`.

(b) Since `f` must return a value of some unknown type `'r`, and since we restrict ourselves to the pure fragment of OCaml, convince yourself that `f k = k v` for some fixed integer `v`.

(c) You cannot see inside `f`. By choosing `'r` cleverly and picking a suitable `k`, write an expression that evaluates to `v`.

QUESTION 15

PRACTICAL · 15 MIN (+ 15 MIN) ●●●○○ **CORE**

An OCaml type `t` is said to be an **ordered type** if it is equipped with a default comparison operator `< : t -> t -> bool` such that `a < b` returns `true` if `a : t` is "smaller than" `b : t`. Many OCaml types — such as `string` and `int` — can be ordered and compared with the `<` operator in the obvious way.

We often want to compare product types, such as `string * int`, building the comparison operators out of the individual atomic types `string` and `int`.

There are two ways of building compound comparators out of atomic comparators. The first is *pairwise ordering*, which compares elements of the pair individually:

$$(x, y) <_p (x', y') \iff x < x' \wedge y < y'$$

The second is *lexicographic ordering*, which orders by the first elements, and if they are equal, by the second element (for an arbitrary number of elements we get the familiar word ordering used in dictionaries):

$$(x, y) <_\ell (x', y') \iff x < x' \vee (x = x' \wedge y < y')$$

(a) Write OCaml functions implementing pairwise and lexicographic ordering for the type of pairs `string * int`.

(b) Your previous answer might have used the default comparison operator `<` for strings and ints. For more flexibility, we might want to pass non-default custom orderings (see next part). Write two higher-order OCaml functions to perform pairwise and lexicographic comparison of values of type `'a * 'b`, where the comparison operators for types `'a` and `'b` are passed as arguments.

(c) Explain how you could use your functions in the previous part, and the higher-order sorting function `insort`, to sort a list of type `(string * (int * string)) list` according to the

following specification:

$$(s_1, (m, s_2)) < (s_1', (n, s_2')) \iff s_1 \leq s_1' \ \wedge \ (m > n \ \vee \ (m = n \ \wedge \ s_2 < s_2'))$$

★ (d) With reference to `("a",(1,"x"))` and `("b",(2,"y"))`, explain why you should not pass this function to a sorting algorithm.

QUESTION 16

PRACTICAL · 20 MIN ●●●○○ CORE

Without using `map`, write a function `map2` such that `map2 f` is equivalent to the composition `map (map f)`. The obvious solution requires declaring two recursive functions. Try to get away with one by exploiting nested pattern-matching.

QUESTION 17

PRACTICAL · 5 MIN ●○○○○ BASIC

The built-in type `option`, shown below, can be viewed as a type of lists having at most one element. Declare an analogue of the function `map` for type `option`.

OCaml
```
type 'a option = None | Some of 'a
```

FMAP

You might be interested in containers other than `list` and `option`. I'd recommend taking a look at Haskell's `fmap`. Note that the `f` stands for *functor*, but these functors are not the same as OCaml's module functors. Instead, the name derives from category theory.

QUESTION 18

CONCEPTUAL · 15 MIN (+ 5 MIN) ●●●○○ CORE

Consider the following program

OCaml
```
let f () =
   let x = 1 in
   let g () = x in
   g

let h = f ();;

let x = 2 in h ()
```

(a) What should the result of this program be?

★ (b) Can you name a language where the result would differ from your answer to part (a)?

(c) One way to implement variables is by looking up their values on the stack. Explain, with reference to the program above, why higher order functions need to be treated specially for this approach to work.

HINT Is `x = 1` still on the stack when `h ()` is evaluated?

QUESTION 19

CONCEPTUAL · 10 MIN (+ 15 MIN) ●●●○○ CORE

Consider the following OCaml program [1]

```OCaml
let int_and_string1: 'a. ('a -> 'a) -> int * string
  = fun arg -> (arg 1, arg "hey!")
```

Now consider the following OCaml program

```OCaml
type poly = { id: 'a. 'a -> 'a }
let int_and_string2: poly -> int * string
  = fun arg -> (arg.id 1, arg.id "hey!")
```

(a) Which of the two programs compiles?

(b) Rewrite the types in second-order logical syntax, something like

$$\forall \alpha \ldots$$

being careful about the bracketing.

★ (c) Can you explain the difference between the two types, and use this difference to explain the difference in behaviour?

HINT You will need to unfold `poly`.

QUESTION 20

PRACTICAL · 15 MIN ●●○○○ BASIC

Recall the making change function of Section 4.6:

```OCaml
let rec change till amt = match till, amt with
  | _, 0  -> [ [] ]
  | [], _ -> []
  | c::till, amt ->
```

[1] This exercise uses examples by Hemant Sai Gouri, though adapted, as the examples in the linked blog post do not compile as written.

```OCaml
      if amt < c then change till amt else
        let rec allc = function
          | [] -> []
          | (cs::css) -> (c::cs) :: allc css
        in allc (change (c::till) (amt-c))
           @ change till amt
```

The function `allc` applies the function "cons a `c`" to every element of a list. Eliminate it by declaring a curried cons function and applying `map`.

---

4 Sequences and laziness

PREREQUISITES Lecture 9 · TIME 1 hour 20 min

The right way to think about lazy data is by the equations it satisfies, not how it is constructed.

QUESTION 21

CONCEPTUAL · 10 MIN ●●○○○ BASIC

Consider the list function `concat`, which concatenates a list of lists to form a single list. Can it be generalised to concatenate a sequence of sequences? What can go wrong?

```OCaml
let rec concat = function
  | []    -> []
  | l::ls -> l @ concat ls
```

QUESTION 22

PRACTICAL · 10 MIN ●○○○○ BASIC

Write an analogue of `map` for sequences.

QUESTION 23

PRACTICAL · 25 MIN ●●●●○ CORE

A *lazy binary tree* is either empty or is a branch containing a label and two lazy binary trees, possibly to infinite depth. Present an OCaml datatype to represent lazy binary trees, along with a function that accepts a lazy binary tree and produces a lazy list that contains all of the tree's labels. The order of the elements in the lazy list does not matter, as long as it contains all potential tree nodes.

QUESTION 24

PRACTICAL · 25 MIN ●●○○○ BASIC

Recall `from`, where `from k` generates the infinite sequence `k, k+1, k+2, ...`

```ocaml
let rec from k = Cons (k, fun () -> from (k+1))
```

(a) Using `from` and your analogue of `map` for sequences, write the sequence of powers of 5: `1, 5, 25, 125, ...`

(b) Now write a generator `g` such that `g 1` represents the same sequence. Do not use `from` or `map`.

(c) Let the sequence `g 1` be seq1 and the sequence `g 5` be seq2.

(i) Write out the first four elements of seq1

(ii) Write out the first four elements of seq2

(iii) Establish a relationship between seq1 and seq2 that does not reference how they were generated (does not mention g)

(d) Consider the sequence

$$25,\ 125,\ 625,\ \ldots$$

which can be characterised in two ways:

(i) the powers of 5, from the third onwards;

(ii) the powers of 5, each multiplied by 25.

How would you modify your answer to part (a) to produce this sequence? How would you modify your answer to part (b)? Which characterisation does each modification correspond to?

(e) Consider the following infinite sequence. It is a geometric progression, potentially familiar from school.

$$T(a) = a, 0.5a, 0.25a, \ldots$$

It is a fact that

$$\sum T(1) = 1 + 0.5 + \ldots = 2$$

Here is the proof: multiplying every term in $T(1)$ by 0.5 gives

$$0.5T(1) = 0.5, 0.25, \ldots = T(0.5)$$

Notice that $0.5T(1)$ is $T(1)$ with its first term removed. Hence we can establish the recursive equation

$$T(1) = 1, 0.5T(1)$$

and hence

$$\sum T(1) = 1 + \sum 0.5T(1) = 1 + 0.5 \sum T(1)$$

and so $\sum T(1) = 2$.

(i) Write $G(n)$ for the geometric progression

$$n, 5n, 25n, \ldots$$

Establish a relationship between $kG(n)$ and $G(kn)$

(ii) Establish a (recursive) equation relating $G(n)$ and $5G(n)$

(iii) Relate the equation in G to your definition of g.

QUESTION 25

PRACTICAL · 10 MIN ●●●○○ CORE

Write the lazy list containing every number of the form $7^a 9^b$, for $a, b \geq 0$: $1, 7, 9, 49, 63, 63, 81, \ldots$ The order does not matter, and duplicates are permitted.

HINT **Question 24 is meant to make this one easy!** Define a family of sequences $H(m)$, where $H(1)$ is the sequence you want. As in question 24, describe two sequences $H'(m)$ and $H''(m)$, each obtained from $H(m)$. Write down an equation relating $H(m)$, $H'(m)$ and $H''(m)$, and turn it into a generator.

FOUNDATIONS OF COMPUTER SCIENCE

Supervision 4

Michaelmas Term 2026 · Michael Lee

WORK DUE	A single PDF, 24 hours before the supervision
EXPECTED TIME	5 hours 45 min (+ 30 min for the optional questions)

1 Sequences and laziness (continued)

PREREQUISITES Lecture 9 · TIME 2 hours 10 min

The right way to think about lazy data is by the equations it satisfies, not how it is constructed.

QUESTION 1

CONCEPTUAL · 15 MIN ●●○○○ BASIC

Let `zip_with_s` be a function which applies a binary operator to corresponding elements of two sequences, so that `zip_with_s (+)` applied to `1, 2, 3, ...` and `5, 6, 7, ...` gives `6, 8, 10, ...` Here is a definition of `zip_with_s`, which you do not need to understand to answer the question:

```OCaml
let rec zip_with_s opr xs ys = match xs, ys with
  | Nil, _ -> Nil
  | _, Nil -> Nil
  | (Cons (x, xf)), (Cons (y, yf)) ->
     Cons (opr x y, fun () -> zip_with_s opr (xf()) (yf()))
```

Here are two definitions of the Fibonacci sequence.

```OCaml
let rec fibs1 = Cons (0, fun () ->
            Cons (1, fun () -> zip_with_s (+) fibs1 (tail fibs1)))

let rec f a b = Cons (a, fun () -> f b (a+b))
let fibs2 = f 0 1
```

Consider the sequences

$$\begin{aligned} \mathcal{F} &= 0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ \ldots \\ \mathcal{F}' &= \quad 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ \ldots \\ \mathcal{F}'' &= \quad\quad 1,\ 2,\ 3,\ 5,\ 8,\ \ldots \end{aligned}$$

(a) Define $\mathcal{F}$, $\mathcal{F}'$, and $\mathcal{F}''$ as instances of some parameterised sequence $F(x, y)$

(b) Establish two equations, one corresponding to `fibs1`, and the other to `fibs2`.

QUESTION 2

PRACTICAL · 25 MIN ●●●●○ CORE

Write the lazy list whose elements are all ordinary lists of zeroes and ones, namely `[]`, `[0]`, `[1]`, `[0;0]`, `[0;1]`, `[1;0]`, `[1;1]`, `[0;0;0]`, ...

QUESTION 3

PRACTICAL · 25 MIN ●●●●○ CORE

(Continuing question 2.) A palindrome is a list that equals its own reverse. Code the lazy list whose elements are all palindromes of 0s and 1s, namely `[]`, `[0]`, `[1]`, `[0;0]`, `[0;0;0]`, `[0;1;0]`, `[1;1]`, `[1;0;1]`, `[1;1;1]`, `[0;0;0;0]`, ... You can use the list reversal function `List.rev`.

QUESTION 4

CONCEPTUAL · 10 MIN ●●●○○ CORE

(a) What is the time complexity of appending two lazy lists?

(b) What is the time complexity of getting the tail of a lazy list?

(c) When might it be desirable to use a lazy list over an eager list?

QUESTION 5

PRACTICAL · 25 MIN ●●●●○ CORE

Memoisation involves storing the results of computation, so as to avoid recomputation. For example, whenever I try and compute

```OCaml
f x
```

I first look up if I have already performed the computation, and if I have, I just look up the result. Otherwise, I compute the result, and then store it in a table for future lookup.

(a) Write a program that tests if memoisation is being performed.

(b) Define `reverseq`, which does not reverse a sequence but instead returns a suspended computation:

```OCaml
let reverseq xq = fun () ->
  let rec reverse xq acc = match xq with
    | Nil         -> acc
    | Cons(x, xf) -> reverse (xf ()) (Cons(x, fun () -> acc))
  in reverse xq Nil
```

Define also a variant of `appendq` whose second argument is a suspended computation, rather than a lazy list:

```OCaml
let rec appendq xq yf = match xq with
  | Nil         -> yf ()
  | Cons(x, xf) -> Cons(x, fun () -> appendq (xf ()) yf)
```

With reference to the following example, explain why memoisation is important for lazy lists.

```OCaml
let rec until n =
  if n = 0 then Nil
  else Cons(n, fun () -> until (n-1))
let xq = Cons(1, fun () -> Nil)
let yq = until 1000
let yqr = reverseq yq
let zq  = appendq xq yqr
let z1 = tail zq
let z2 = tail zq
```

QUESTION 6

CONCEPTUAL · 30 MIN ●●●○○ CORE

For simplicity, consider the pure, terminating subset of OCaml that only has expressions and values (the question generalises to OCaml programs). Recall that OCaml expressions evaluate to values. Let A and B be OCaml types. Let $t_1 : A \to B$ and $t_2 : A$ be two OCaml programs. Let t_1 evaluate to v_1, written $t_1 \rightsquigarrow v_1$ and $t_2 \rightsquigarrow v_2$. Further, since v_1 has a function type, it must have shape $\text{fun}\, x \to t$. Function application results in substitution, written $t[t'/x]$ read "t with t' substituted for x". Concretely, if $t = x + 1$, and $t' = 2$, then

$$t[t'/x] = (x+1)[2/x] = 2 + 1$$

Consider evaluating the application $t_1\, t_2$. Consider the following two evaluation strategies

1. Completely evaluate the function, completely evaluate its argument, and then perform substitution

$$\begin{array}{rcll} t_1\, t_2 & \rightsquigarrow & (\text{fun}\, x \to t)\, t_2 & \text{(reduce the function)} \\ & \rightsquigarrow & (\text{fun}\, x \to t)\, v_2 & \text{(reduce the argument)} \\ & \rightsquigarrow & t[v_2/x] & \text{(perform substitution)} \end{array}$$

2. Completely evaluate the function, and then perform substitution without evaluating the argument

$$\begin{array}{rcll} t_1\, t_2 & \rightsquigarrow & (\text{fun}\, x \to t)\, t_2 & \text{(reduce the function)} \\ & \rightsquigarrow & t[t_2/x] & \text{(perform substitution)} \end{array}$$

Note that the choice of strategy is a **language** choice, not a user choice. Different languages make different choices.

(a) Which of the evaluation strategies is call-by-value, and which is call-by-name?

(b) Write a program that evaluates differently under the two evaluation strategies (even if they eventually agree on the answer, they take different steps to get there), and show how they evaluate in each case.

(c) Widen the subset of OCaml to include a non-terminating program ω. We can write ω in OCaml as such

```OCaml
let rec omega () = omega () in omega ()
```

Write a program where call-by-value and call-by-name will exhibit different behaviour.

(d) Invent an approach to simulate the call-by-name evaluation strategy in a language that has chosen the call-by-value evaluation strategy.

(e) Compare your approach to the transition from lists to lazy lists. What can you say about OCaml's evaluation strategy, and what can you say about the design of lazy lists?

---

2 Queues and search strategies

PREREQUISITES Lecture 10 · TIME 1 hour 45 min

A careful look at building persistent amortised data structures, and a quick look at search strategies.

QUESTION 7

CONCEPTUAL · 20 MIN ●●●○○ CORE

Suppose that we have an implementation of queues, based on binary trees, such that all operations have worst-case time complexity $O(\log n)$.

Outline the advantages and drawbacks of such an implementation compared with the efficient functional queues presented in Section 10.5.

QUESTION 8

CONCEPTUAL · 35 MIN ●●●●● ADVANCED

This question is inspired by the introductory sections of Anton Lorenzen's *First Order Laziness*, which in turn is inspired by Chris Okasaki's PhD thesis, *Purely Functional Data Structures*.

The version of a queue from the notes performs normalisation whenever `hds` is empty. Another alternative is to perform normalisation whenever `hds` is shorter than `tls`.

(a) Implement this variant queue

(b) Assume we have a queue `q`, where the length of `hds` is the same as the length of `tls`, and call this length `n`. Analyse the time complexity of the following program.

```OCaml
for i = 1 to n do
  ignore (deq q)
done
```

MOTIVATION: CONCURRENT SYSTEMS

Note that this example, while contrived, reflects a very real use case: in concurrent and distributed systems, one often faces cases with multiple readers of the same data structure. This program simulates multiple reader threads of the same queue.

(c) In the earlier example, each `deq` potentially involves a normalisation:

```OCaml
append hds (reverse tls)
```

Now consider using lazy lists for `hds` and `tls` (recall the definitions of `reverseq` and `appendq` from question 5)

```OCaml
appendq hds (reverseq tls)
```

might be more efficient in this case. What about without memoisation?

QUESTION 9

CONCEPTUAL · 5 MIN ●○○○○ BASIC

The traditional way to implement queues uses a fixed-length array. Two indices into the array indicate the start and end of the queue, which wraps around from the end of the array to the start. How appropriate is such a data structure for implementing breadth-first search?

QUESTION 10

CONCEPTUAL · 5 MIN ●○○○○ BASIC

Why is iterative deepening inappropriate if $b \approx 1$, where b is the branching factor? What search strategy would make more sense in this case?

QUESTION 11

CONCEPTUAL · 10 MIN ●●○○○ BASIC

An interesting variation on the data structures seen in the lectures is a *deque*, or *double-ended queue*. A deque supports efficient addition and removal of elements on both ends (either its front or back). Suggest a suitable implementation of deques, justifying your decision.

QUESTION 12

PRACTICAL · 30 MIN ●●●●○ CORE

Implement deques as an abstract data type in OCaml (including a type declaration and suitable operations). Estimate the amortised complexity of your solution.

PREREQUISITES Lecture 11 · TIME 1 hour 50 min (+ 30 min)

3 Elements of procedural programming

OCaml has references, assignment and `while` loops. These are important tools in a programmer's arsenal, but they come with high costs: this section is about understanding how to use them, the price one pays for using them, and how to use them well.

QUESTION 13

CONCEPTUAL · 20 MIN ●●●○○ CORE

(a) Assume `e` and `e'` are arbitrary OCaml expressions of type `int`. Is it true that `e + e'` is always the same as `e' + e`? Either explain why it is true, or write down `e` and `e'` that will exhibit different behaviour.

(b) What if `e` and `e'` are pure expressions?

QUESTION 14

CONCEPTUAL · 15 MIN ●●○○○ BASIC

Comment, with examples, on the differences between an `int ref list` and an `int list ref`. How would you convert between the two?

QUESTION 15

PRACTICAL · 10 MIN ●○○○○ BASIC

Write a version of function `power` (Section 1.6) using `while` instead of recursion.

QUESTION 16

PRACTICAL · 10 MIN (+ 20 MIN) ●●●○○ CORE

This question walks you through the creation of a memoised power function, using OCaml's `Hashtbl` module. The important functions are `Hashtbl.create`, `Hashtbl.find_opt` and `Hashtbl.replace`.

(a) Write `pow : int -> int -> int`, which consults a table before computing and stores what it computes.

(b) Identify two bad places to create the table, and explain why your choice is better than either of them.

(c) Is `pow 2 100` any faster if you have already computed `pow 2 99`?

(d) Is `pow` pure?

★ (e) Assume you have access to a constant-time prime factorisation algorithm,

```
val factorise: int -> (int * int) list
```

For example, `factorise 100 = [(2, 2); (5, 2)]`. You can also assume a gcd algorithm

```
val gcd: int list -> int * int list
```

For example, `gcd [2;4] = (2, [1;2])`, and `gcd [] = (0, [])`. Can you think of a way to make your solution even faster?

HINT If `pow 2 100` is in the table, how much time should it take to compute `pow 4 50`?

QUESTION 17

PRACTICAL · 5 MIN (+ 10 MIN) ●○○○○ BASIC

(a) Write a function to exchange the values of two `t refs`, `xr` and `yr`.

★ (b) If `t = int`, and you may assume `xr != yr`, can you improve your solution?

QUESTION 18

PRACTICAL · 25 MIN ●●●○○ CORE

Arrays of multiple dimensions are represented in OCaml by arrays of arrays. Write functions to (a) create an $n \times n$ identity matrix, given n, and (b) to transpose an $m \times n$ matrix. Identity matrices have the following form:

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

QUESTION 19

CONCEPTUAL · 25 MIN ●●●●● ADVANCED

OCaml enforces abstraction barriers through its powerful module system. Specifically, its module system offers *static sealing*. Consider the following module for the natural numbers

OCaml

```
module type NAT = sig
  type t
  val zero : t
  val succ : t -> t
end

module Nat: NAT = struct
  type t = int
  let zero = 0
  let succ n = n+1
end
```

The module type ascription `Nat : NAT` seals the module, preventing code from outside the module from peeking into it. This creates a form of unforgeability: the only way to get an element of type `t` is to ask the module for one. We cannot try to forge a value of type `t`, the following example gives a type error:

```OCaml
Nat.succ 0
```

Static sealing is a feature offered by the language: attempts to break it are caught by the compiler. If the language does not implement static sealing, it is sometimes possible to build *dynamic* sealing, which offers similar guarantees, but only when the program runs (not when it is compiled).

The idea behind dynamic sealing is simple: there is a type of keys, which cannot be forged, and a type of sealed values. The user can mint new keys. A key can be used to seal values, creating a sealed value. A key can also be used in an attempt to unseal a sealed value. If the key was used to seal the value, then unsealing succeeds, returning `Some v`. Otherwise, it fails, returning `None`.

As described by Filinski in Remark 4.30 of his PhD thesis *Controlling Effects*, and translated into OCaml by Dr. Jeremy Yallop, we can implement dynamic sealing in OCaml.

```OCaml
module type SEAL =
sig
  type sealed
  type 'a key
  val make_key : unit -> 'a key
  val seal : 'a -> 'a key -> sealed
  val unseal : 'a key -> sealed -> 'a option
end
module Seal: SEAL = struct
  type sealed = unit -> unit
  type 'a key = 'a option ref
  let make_key () = ref None
  let seal v r = fun () -> r := Some v
  let unseal r f = r := None; f (); !r
end
```

(a) How have we implemented keys?

(b) Why are keys unforgeable?

(c) Explain how `unseal k (seal v k)` succeeds

(d) Explain how `unseal k (seal v k')` fails when `k` and `k'` are different