

[CSE301 / Lecture 0]
An introduction to functional programming

Noam Zeilberger

Ecole Polytechnique

4 September 2023

Course staff

Lectures and Labs: me [noamz.org]

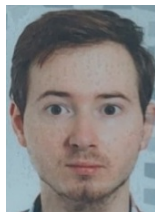
Labs:



Jill-Jênn Vie

jill-jenn.net

Inria researcher, Soda team
(only today)



Théo Boury

PhD student, AMIBio team
(from next week)

What is functional programming?

Hard to give a precise definition, but a rough approximation is that functional programming is *a style of programming that emphasizes **function application** and **function composition**.*

Function application

“The act of evaluating a function on some argument.”

E.g., apply $f(x) = \sqrt{x}$ to 2, obtaining $f(2) \approx 1.41421$.

Depending on the compiler, application may be implemented using lower-level operations like pushing arguments onto a stack, etc. – although sometimes we need to know these details, usually we can treat function application as a higher-level abstraction.

Function composition

“The act of combining two or more fns. to define a new function.”

E.g., given $f(x) = \sqrt{x}$, $g(x) = \sin x$, $h(x) = e^x$, define

$$i(x) = h(f(x) + g(x))$$

Observe we can compose both “in sequence” and “in parallel”.

Again, a compiler may need to make additional choices (e.g., store $f(x)$ before computing $g(x)$? vice versa? multicore?), but the functional notation nicely captures just the logical dependencies.

What makes a programming language “functional”?

Possible to program in the functional style in almost any language, but a *functional programming language* makes it easier.

Typically, by including at least some of the following features:

pattern-matching **higher-order functions** **rigorous typing**

(This is not an “official” list. But you will hopefully come to appreciate why these three features are especially useful.)

A few examples of languages with all these features:

Haskell, OCaml, Coq, Agda, Rust, Lean, ...

(Quick poll: who has used any of these languages?)

Why learn functional programming?

FP had a reputation as “academic” for a long time, but Haskell and OCaml have been used in industry for at least two decades, and FP concepts are increasingly going mainstream.

Some practical benefits of FP:

- Powerful notations inspired by mathematics and logic
- Better control over “side-effects” of functions
- In principle, easier to parallelize

Overall, FP simplifies the task of going from an abstract description of a problem to an efficient and reliable implementation in code.

... But also: *it's beautiful!*

An example

$partition :: (a \rightarrow Bool) \rightarrow [a] \rightarrow ([a], [a])$

$partition\ p\ [] = ([], [])$

$partition\ p\ (x : xs) = \mathbf{if}\ p\ x\ \mathbf{then}\ (x : ts, fs)\ \mathbf{else}\ (ts, x : fs)$

where

$(ts, fs) = partition\ p\ xs$

$qsort :: Ord\ a \Rightarrow [a] \rightarrow [a]$

$qsort\ [] = []$

$qsort\ (x : xs) = qsort\ left\ \mathbf{++}\ [x]\ \mathbf{++}\ qsort\ right$

where

$(left, right) = partition\ (\backslash y \rightarrow y < x)\ xs$

A brief (pre-)history

1920s-30s: Alonzo Church and his students Kleene and Rosser develop λ -calculus.

1937: Alan Turing proves equivalence between TM-computability and λ -definability.

late 1950s: John McCarthy develops the LISP language.

mid 1960s: Peter Landin promotes λ -calculus as a conceptual tool for reasoning about programming languages.

1970s: striking connections between programming, logic, & math!

Unwinding the Curry-Howard-Lambek correspondence

In the tumultuous 1970s (and late '60s):

- Dana Scott invents *domain theory*
- Jean-Yves Girard & John Reynolds both independently discover the *polymorphic λ -calculus*
- J. Roger Hindley and Robin Milner both independently discover an algorithm for *polymorphic type inference*
- Per Martin-Löf introduces *dependent type theory*
- Joachim Lambek's work on *cartesian closed categories*, building on Bill Lawvere's earlier work on categorical logic, as well as Lambek's own older work in mathematical linguistics

More recent history

also in the 1970s: Guy Steele & Gerald Sussman develop Scheme and write “Lambda: The Ultimate” series of papers

also also in the 1970s: Robin Milner and others develop ML

1980s: ML evolves into Standard ML and Caml (later OCaml)

1987: an international committee starts work on Haskell

1989: first release of the Coq proof assistant

1992: Phil Wadler’s “Monads for functional programming”

1996: OCaml developed by Xavier Leroy, Jérôme Vouillon, et cie

2007: first release of Agda proof assistant, written in Haskell

2021: “LAMBDA: The ultimate Excel worksheet function”

Coincidentally this week...



Mon 4 - Sat 9 September 2023
Seattle, Washington, United States

The 28th ACM SIGPLAN International Conference on Functional Programming will be in Seattle, WA, USA on 4-9 September 2023.

ICFP's main research paper track will be on September 5-7 (Tuesday-Thursday), with workshops, tutorials, and other events on September 4 and 8-9 (Monday, and Friday-Saturday).

[Registration](#) is now open!

Keynotes



TypeScript: Static types for JavaScript

Anders Hejlsberg



Programming for the planet

Anil Madhavapeddy



As low-level as possible, but no lower

Andreas Rossberg



The Cogent and Dargent approach to facilitating verified software development

Christine Rizkallah

Why Haskell for this course?

An elegant language with a rich ecosystem. (So is OCaml.)

Haskell is a **pure** & **lazy** functional programming language:

- Purity forces you to think more rigorously about side-effects. (Although monads are not the ultimate answer!)
- In retrospect, laziness was probably a bad idea, but at least it is an interesting one! (We will study it, but not emphasize it.)

Ultimately, we will just use Haskell as an *intellectual tool* for learning about functional programming. (Although you may still eventually find it useful in your practical programming!)

Course practicalities

Use Moodle for handing in assignments, including today's lab.

Also, please use the Q & A forum on Moodle.

The course webpage contains more information:

<https://www.lix.polytechnique.fr/Labo/Noam.ZEILBERGER/teaching/CSE301/>

(Let's go over it now.)