Lenguajes de Programación

Laziness in Haskell

Ismael Figueroa

dcc
CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Adaptación contenido
profesor Federico Olmedo

Haskell

An advanced, purely functional programming language

Features

Statically typed

Every expression in Haskell has a type which is determined at compile time. All the types composed together by function application have to match up. If they don't, the program will be rejected by the compiler. Types become not only a form of guarantee, but a language for expressing the construction of programs.

Purely functional

Every function in Haskell is a function in the mathematical sense (i.e., "pure"). Even side-effecting IO operations are but a description of what to do, produced by pure code. There are no statements or instructions, only expressions which cannot mutate variables (local or global) nor access state like time or random numbers.

www.haskell.org

Type inference

You don't have to explicitly write out every type in a Haskell program. Types will be inferred by unifying every type bidirectionally. However, you can write out types if you choose, or ask the compiler to write them for you for handy documentation.

Click to expand

Concurrent

Haskell lends itself well to concurrent programming due to its explicit handling of effects. Its flagship compiler, GHC, comes with a high-performance parallel garbage collector and light-weight concurrency library containing a number of useful concurrency primitives and abstractions.

Click to expand

Lazy

Functions don't evaluate their arguments. This means that programs can compose together very well, with the ability to write control constructs (such as if/else) just by writing normal functions. The purity of Haskell code makes it easy to fuse chains of functions together, allowing for performance benefits.

Click to expand

Packages

Open source contribution to Haskell is very active with a wide range of packages available on the public package servers.

Click to expand

Haskell

An advanced, purely functional programming language

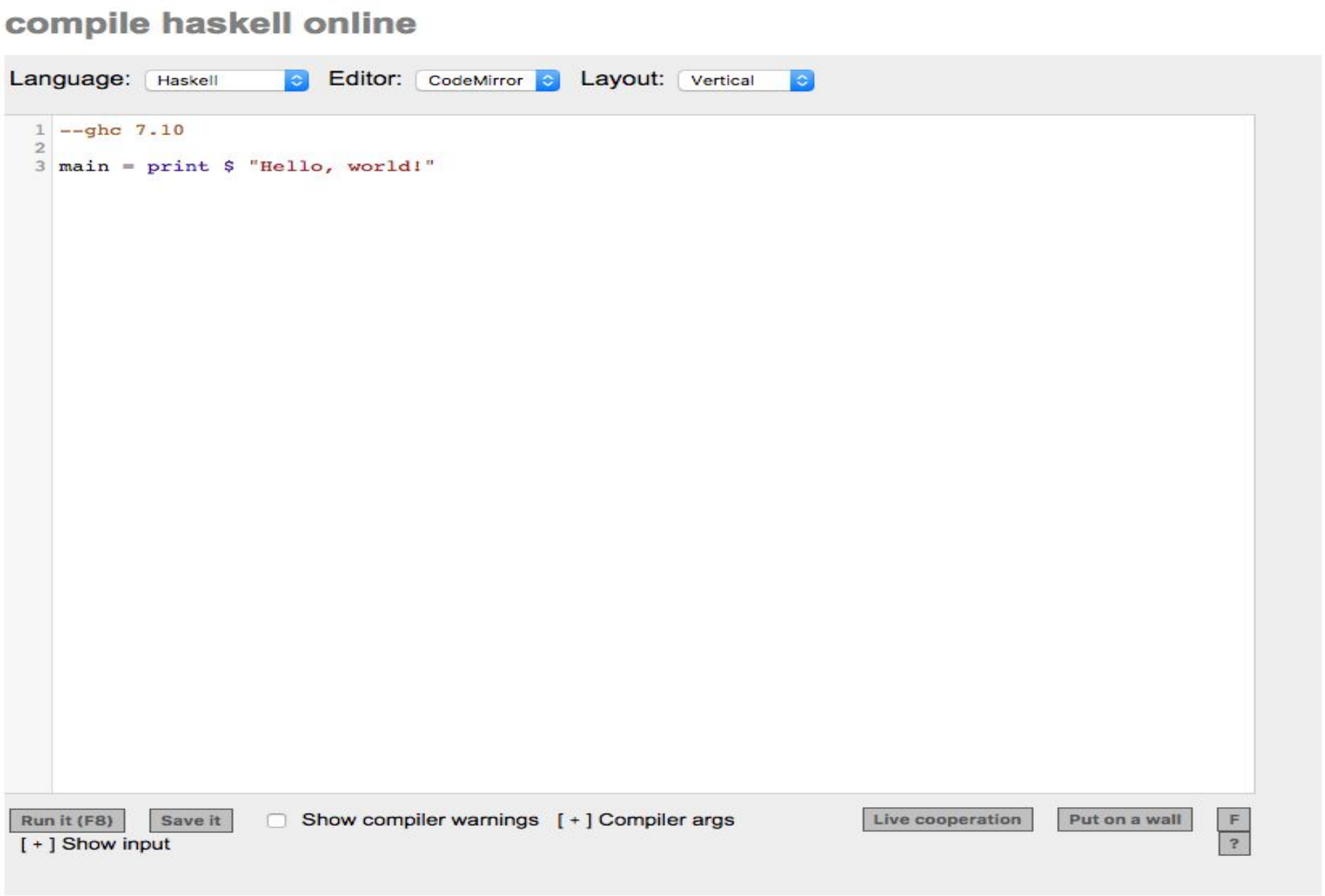
Haskell Brooks Curry *was an American mathematician and logician. Curry is best known for his work in combinatory logic; while the initial concept of combinatory logic was based on a single paper by Moses Schönfinkel, much of the development was done by Curry. Curry is also known for Curry's paradox and the* **Curry–Howard correspondence**. *There are three programming languages named after him, Haskell, Brook and Curry, as well as the concept of currying, a technique used for transforming functions in mathematics and computer science.*

Haskell Curry (1900 –1982)

https://en.wikipedia.org/wiki/Haskell_Curry

Setting up Haskell (remote interpretation)

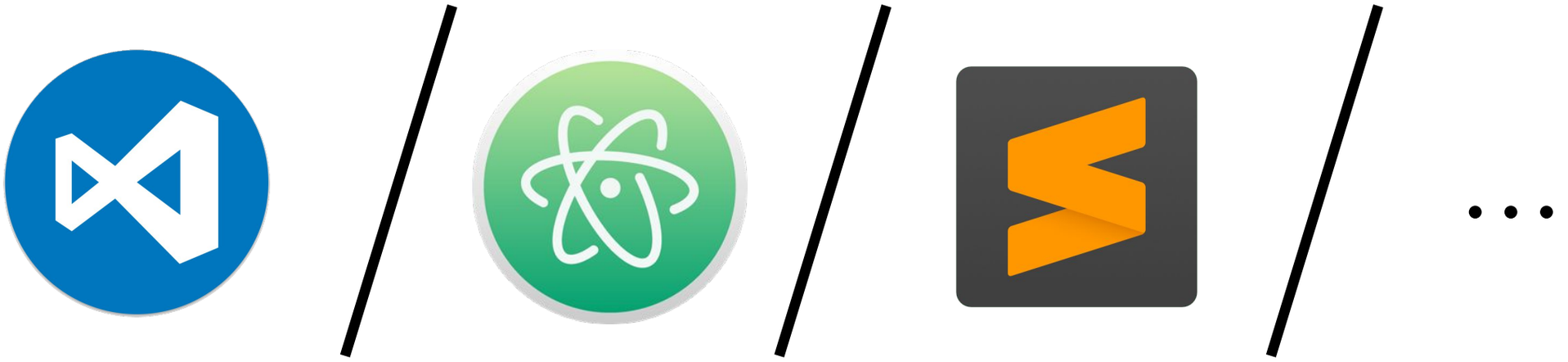
http://rextester.com/l/haskell_online_compiler



Setting up Haskell (local interpretation)

https://www.haskell.org/downloads/

+



Your preferred editor / IDE

A brief tour on Haskell

Infinite data structures

Define function

mapRep :: (a -> a) -> a -> [a]

such that

mapRep f x = [x, f x, f^2 x, f^3 x, f^4 x, …]

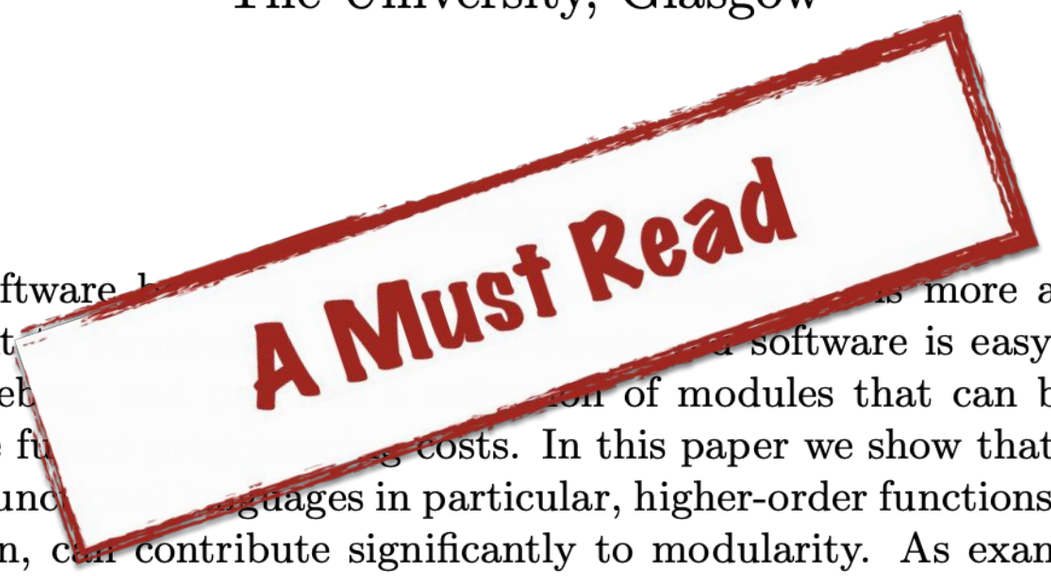
Using our arithmetical language define function my-S

my-S(0) = 0 my-S(n+1) = n+1 + my-S(n)

that sums up the first n natural numbers.

Why Functional Programming Matters

John Hughes
The University, Glasgow



As software b[illegible] more and more important [illegible] software is easy to write and to deb[illegible] of modules that can be reused to reduce fu[illegible] costs. In this paper we show that two features of func[illegible]guages in particular, higher-order functions and lazy evaluation, can contribute significantly to modularity. As examples, we manipulate lists and trees, program several numerical algorithms, and implement the alpha-beta heuristic (an algorithm from Artificial Intelligence used in game-playing programs). We conclude that since modularity is the key to successful programming, functional programming offers important advantages for software development.

[Download]

Our interpreter in Haskell

Comparison between Racket and Haskell

Racket

- Eager evaluation
- Dynamically type-checked
- (Mostly) uncurrified functions
- Prefix parenthetical syntax

Haskell

- Lazy evaluation
  - Infinite data structures
- Statically type-checked
- (Mostly) currified functions
- More natural syntax

Si te interesa aprender más sobre Haskell,

y sobre la programación funcional en general,

Programación Funcional

Curso electivo

Lecture material

Bibliography

- Programming Languages: Application and Interpretation (1st Edition)
  Shriram Krishnamurthi [Download]
  Chapter 7

Source code

- A potpourri of what we did in class [Download]

Optional bibliography to learn more about Haskell

- Learn You a Haskell: (funny) online book