Haskell

K31 Compilers, Spring 2023

K. Chousos, K. Kordolaimis, P. Seitanidis, A. Tsitsoli

Introduction

Raison d'être

- Haskell was created by a committee that was formed in the FPCA conference of 1987.
- Its purpose was to unite the functional programming world by being a language that is:
  - Accessible (open-source)
  - Stable
  - Disruptive
  - Useful for teaching, research, as well for real-world applications

Where it is used

Haskell is used in a plethora of fields and industries to solve a variety of problems. For example:

- The Cardano blockchain platform, its protocol and its smart contract infrastructure
- In Facebook for filtering spam.
- Hasura, an open-source real-time GraphQL engine is written in Haskell.

Features

Laziness

- A very distinctive feature of Haskell is *Lazy evaluation*.
- Its usefulness lies in the fact that it delays the computation of an expression until the value of the computation is needed.
- The efficiency of this operation provides the reduction of unnecessary calculations alongside the saving of time.
- The downside of this process is that the usage of memory can be unpredictable.

Lambdas

Lambda expressions are a way of defining an anonymous function, helping developers define functions “on the fly” and passing them directly as arguments to other functions.

Syntax of lambda expressions:

```
\arg1 arg2 ... argN -> body
-- Example: Lambda that adds two numbers
\x y -> x + y
```

Monads

- Monads allow you to express complex computations in a modular way, granting the ability to represent computations as **sequences of steps**.
- It is a type class that defines two operations: `return` and `>>=`.
  - The return operation takes a value and puts it in a monad, lifting a **pure value** into a **monadic context**.
  - The `>>=` operation is employed to connect monadic computations together.

Monads

How I/O operations are preformed in Haskell:

```haskell
main :: IO ()
main = do
    putStrLn "What is your name ?"
    name <- getLine
    putStrLn ("Hello, " ++ name ++ "!")
```

Memory management

- Haskell's ghc implementation uses a parallel generational-copying garbage collector. That means that all memory allocations and releases happen in the background automatically.
- In every step, Haskell's GC searches for any *live data*: Data pointed to by the *roots* (the stack and any global variables).
- The live data found are copied to a new heap; Any *dead* data is discarded.

Prelude

Haskell's API is represented by its *Prelude*, a structure which holds a set of standard definitions and is included automatically in all of Haskell's modules. The definitions include:

- Functions: `map`, `filter`, `read`
- Data Types: `Int`, `Bool`, `Char`
- Type Classes: `Eq`, `Read`, `Num`

Example code

Factorial

- Here's a small program that computes the factorial of a given number recursively.

```
factorial :: Int -> Int
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

- Here's a program that does the same thing by leveraging *lambdas*.

```
factorial n = product [1..n]
```

Thank you!