Functional programming, Seminar No. 1

Danya Rogozin
Institute for Information Transmission Problems, RAS
Serokell OÜ

Higher School of Economics
The Department of Computer Science

General words on Haskell and History

Intro

- The language is named after Haskell Curry, an American logician
- The first implementation: 1990
- The language standard: Haskell2010
- Default compiler: Glasgow Haskell compiler
- Haskell is a strongly-typed, polymorphic, and purely functional programming language

Intro

- The language is named after Haskell Curry, an American logician
- The first implementation: 1990
- The language standard: Haskell2010
- Default compiler: Glasgow Haskell compiler
- Haskell is a strongly-typed, polymorphic, and purely functional programming language
- This course is quite introductory.

Intro

- The language is named after Haskell Curry, an American logician
- The first implementation: 1990
- The language standard: Haskell2010
- Default compiler: Glasgow Haskell compiler
- Haskell is a strongly-typed, polymorphic, and purely functional programming language
- This course is quite introductory.
- Vox populi:

Chris Burnor
@chrisburnor

Is Haskell the Rick and Morty of programming languages? 🤔 twitter.com/thejameskyle/s...

Lambda calculus and type theory. Incomplete and Utter History of Functional Programming

- At the end of the 1920-s, Alonzo Church proposed an alternative approach to the foundations of mathematics where the notion of a function is the primitive one. The lambda calculus is a formal system that describes arbitrary abstract functions.

Lambda calculus and type theory. Incomplete and Utter History of Functional Programming

- At the end of the 1920-s, Alonzo Church proposed an alternative approach to the foundations of mathematics where the notion of a function is the primitive one. The lambda calculus is a formal system that describes arbitrary abstract functions.
- Moreover, Church used the lambda calculus to show that Peano arithmetic is undecidable.

Lambda calculus and type theory. Incomplete and Utter History of Functional Programming

- At the end of the 1920-s, Alonzo Church proposed an alternative approach to the foundations of mathematics where the notion of a function is the primitive one. The lambda calculus is a formal system that describes arbitrary abstract functions.
- Moreover, Church used the lambda calculus to show that Peano arithmetic is undecidable.
- Kleene and Rosser showed that the initial version of the lambda calculus is inconsistent. Initially, the idea of typing was the instrument that would allow us to avoid paradoxes.

Lambda calculus and type theory. Incomplete and Utter History of Functional Programming

- At the end of the 1920-s, Alonzo Church proposed an alternative approach to the foundations of mathematics where the notion of a function is the primitive one. The lambda calculus is a formal system that describes arbitrary abstract functions.
- Moreover, Church used the lambda calculus to show that Peano arithmetic is undecidable.
- Kleene and Rosser showed that the initial version of the lambda calculus is inconsistent. Initially, the idea of typing was the instrument that would allow us to avoid paradoxes.
- The first system of typed the lambda calculus is a hybrid from the lambda calculus and type theory developed by Bertrand Russell and Alfred North Whitehead (1910-s).

Lambda calculus and type theory. Historical notes

- After Church's works, type theory as the branch of λ calculus and combinatory logic was developed by Haskell Curry and William Howard within the context of proof theory (1950-1960-s)

Lambda calculus and type theory. Historical notes

- After Church's works, type theory as the branch of λ calculus and combinatory logic was developed by Haskell Curry and William Howard within the context of proof theory (1950-1960-s)
- Polymorphic lambda calculus (John Reynolds and Jean-Yves Girard (1970-s))

Lambda calculus and type theory. Historical notes

- After Church's works, type theory as the branch of λ calculus and combinatory logic was developed by Haskell Curry and William Howard within the context of proof theory (1950-1960-s)
- Polymorphic lambda calculus (John Reynolds and Jean-Yves Girard (1970-s))
- Polymorphic type inference (Roger Hindley, Robin Milner and Luis Damas (1970-1980-s))

Lambda calculus and type theory. Historical notes

- After Church's works, type theory as the branch of λ calculus and combinatory logic was developed by Haskell Curry and William Howard within the context of proof theory (1950-1960-s)
- Polymorphic lambda calculus (John Reynolds and Jean-Yves Girard (1970-s))
- Polymorphic type inference (Roger Hindley, Robin Milner and Luis Damas (1970-1980-s))
- ML: the very first language with polymorphic inferred type system (Robin Milner, 1973)

Lambda calculus and type theory. Historical notes

- After Church's works, type theory as the branch of λ calculus and combinatory logic was developed by Haskell Curry and William Howard within the context of proof theory (1950-1960-s)
- Polymorphic lambda calculus (John Reynolds and Jean-Yves Girard (1970-s))
- Polymorphic type inference (Roger Hindley, Robin Milner and Luis Damas (1970-1980-s))
- ML: the very first language with polymorphic inferred type system (Robin Milner, 1973)
- The language Haskell appeared at the beginning of 1990-s. Haskell desinged by Simon Peyton Jones, Philip Wadler, and others

Functional programming and its foundations

The lambda calculus establishes the foundations for functional programming in the same manner as the von Neumann principles are the root of imperative programming.

Functional programming and its foundations

The lambda calculus establishes the foundations for functional programming in the same manner as the von Neumann principles are the root of imperative programming.

- We have no assignment in imperative languages. Variables are nullary constant functions rather than so-called boxes.
- Hence, we have no states
- We use recursion instead of loops
- Pattern-matching

What are types needed for?

According to Benjamin Pierce

A type system is a tractable syntactic method for proving the absence of certain program behaviours by classifying phrases according to the kinds of values they compute.

What are types needed for?

According to Benjamin Pierce

A type system is a tractable syntactic method for proving the absence of certain program behaviours by classifying phrases according to the kinds of values they compute.

It's about

- A partial specification
- Type preserving
- Type checking allows one to catch simple errors
- Type inference
- Etc

A landscape of typing from a bird's eye view

We may classify possible ways of typing as follows

- Static and dynamic typing
  - C, C++, Java, Haskell, etc
  - JavaScript, Ruby, PHP, etc
- Implicit and explicit typing
  - JavaScript, Ruby, PHP, etc
  - C++, Java, etc
- Inferred typing
  - Haskell, Standard ML, Ocaml, Idris, etc

Ecosystem

The Haskell Platform installation

There are several ways to install the Haskell platform on Mac:

- Download the `.pkg` file and install the corresponding package

The Haskell Platform installation

There are several ways to install the Haskell platform on Mac:

- Download the `.pkg` file and install the corresponding package
- Run the script

```
curl -sSL https://get.haskellstack.org/ | sh
```

The Haskell Platform installation

There are several ways to install the Haskell platform on Mac:

- Download the `.pkg` file and install the corresponding package
- Run the script

```
curl -sSL https://get.haskellstack.org/ | sh
```

- Install ghc, stack, and cabal using Homebrew

The Haskell Platform installation

There are several ways to install the Haskell platform on Mac:

- Download the `.pkg` file and install the corresponding package
- Run the script

```
curl -sSL https://get.haskellstack.org/ | sh
```

- Install ghc, stack, and cabal using Homebrew

Choose any way you prefer. All these ways are equivalent to each other.

The Haskell Platform installation

There are several ways to install the Haskell platform on Mac:

- Download the `.pkg` file and install the corresponding package
- Run the script

```
curl -sSL https://get.haskellstack.org/ | sh
```

- Install ghc, stack, and cabal using Homebrew

Choose any way you prefer. All these ways are equivalent to each other.

I'm a Mac user, but I believe that you'll manage to install the Haskell Platform on NixOs/Windows/Linux/etc quite quickly.

GHC

- GHC is a default Haskell compiler.
- GHC is an open-source project. Don’t hesitate to contribute!
- GHC is mostly implemented on Haskell.
- GHC is developed under the GHC Steering committee control.

GHC

- GHC is a default Haskell compiler.
- GHC is an open-source project. Don't hesitate to contribute!
- GHC is mostly implemented on Haskell.
- GHC is developed under the GHC Steering committee control.
- Very roughly, compiling pipeline is arranged as follows:

  parsing $\Rightarrow$ compile-time (type-checking mostly) $\Rightarrow$ runtime (program execution)

GHCi

- GHCi is a Haskell interpreter based on GHC.
- One may run GHCi with the command `ghci`.
- You may play with GHCi as a calculator, the ordinary arithmetic operators are usual
- You may also have a look at the GHCi chapter in the GHC User's Guide to get familiar with GHCi closely.

```
MacBook-Pro-Daniel:~ suedehead$ ghci
GHCi, version 8.8.1: https://www.haskell.org/ghc/  :? for help
Prelude>
```

Cabal

- Cabal is a system of library and dependency management
- A `.cabal` file describes the version of a package and its dependencies
- Cabal is also a packaging tool
- Cabal used to cause dependency hell

Stack

- Stack is a *mainstream* cross-platform build tool for Haskell projects
- Stack is about

Stack

- Stack is a *mainstream* cross-platform build tool for Haskell projects
- Stack is about
  - installation of required packages and the latest GHC (and their more concrete versions),
  - building, execution, and testing
  - creating an isolated location.
  - Builds are reproducible

Snapshots

- A *snapshot* is a curated package set used by Stack

Snapshots

- A *snapshot* is a curated package set used by Stack
- Stackage is a stable repository that stores snapshots

Snapshots

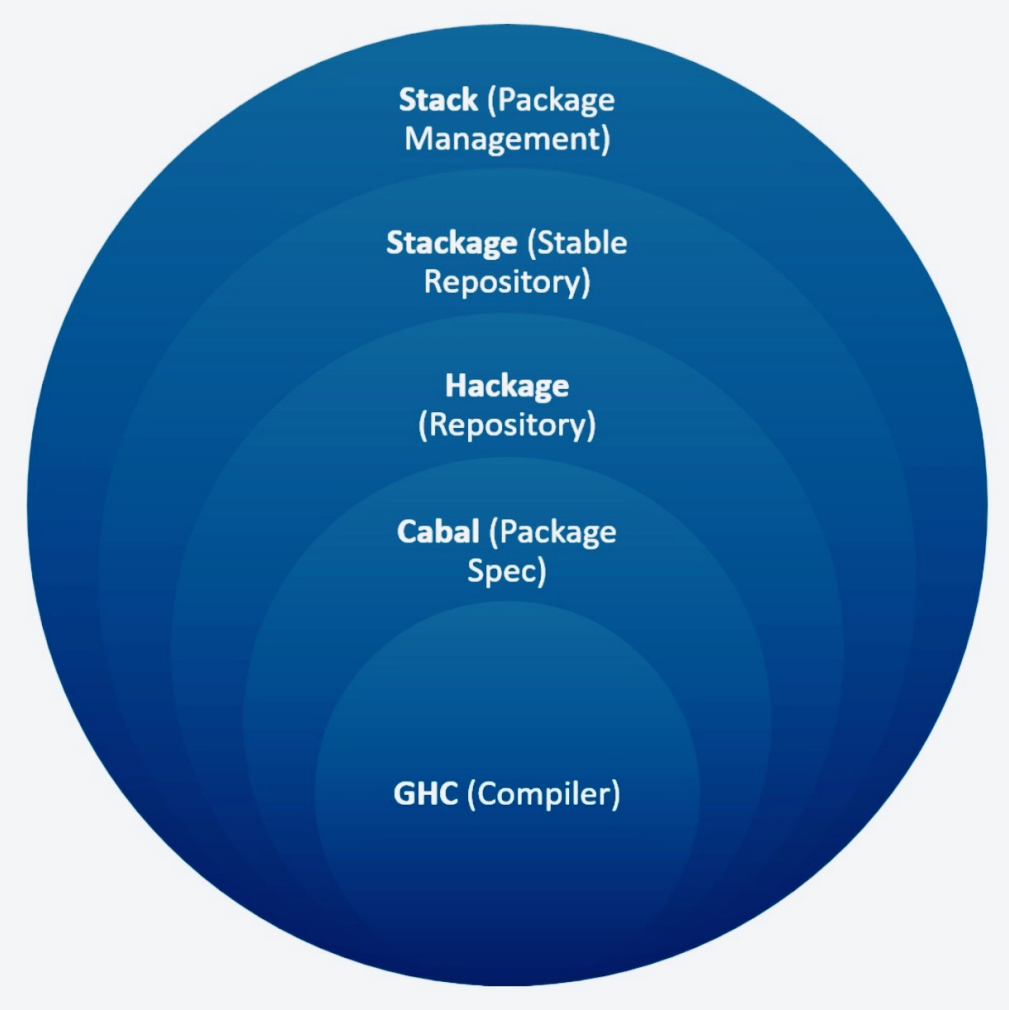
- A *snapshot* is a curated package set used by Stack
- Stackage is a stable repository that stores snapshots
- A *resolver* is a reference to a required snapshot

Snapshots

- A *snapshot* is a curated package set used by Stack
- Stackage is a stable repository that stores snapshots
- A *resolver* is a reference to a required snapshot
- A screenshot from Stackage:

Ecosystem encapsulation

The Haskell ecosystem encapsulation might be described as the sequence of the following inclusions:



Creating a Haskell project with Stack

- Figure out how to call your project and run the script `stack new <projectname>`
- You will see the following story after the command `tree .` in the project directory:

Creating a Haskell project with Stack

- Figure out how to call your project and run the script `stack new <projectname>`
- You will see the following story after the command `tree .` in the project directory:

```
MacBook-Pro-Daniel:myFirstProject suedehead$ tree .
.
├── ChangeLog.md
├── LICENSE
├── README.md
├── Setup.hs
├── app
│   └── Main.hs
├── myFirstProject.cabal
├── package.yaml
├── src
│   └── Lib.hs
├── stack.yaml
└── test
    └── Spec.hs

3 directories, 10 files
```

`stack.yaml`

Let us discuss on how dependency files look like. First of all, we observe the `stack.yaml` file:

stack.yaml

Let us discuss on how dependency files look like. First of all, we observe the `stack.yaml` file:

```
resolver: lts-14.19

# User packages to be built.
# Various formats can be used as shown in the example below.
#
# packages:
# - some-directory
# - https://example.com/foo/bar/baz-0.0.2.tar.gz
#   subdirs:
#   - auto-update
#   - wai
packages:
- .

# extra-deps:
# - acme-missiles-0.3
# - git: https://github.com/commercialhaskell/stack.git
#   commit: e7b331f14bcffb8367cd58fbfc8b40ec7642100a
#
# extra-deps: []
```

Cabal file

As we told above, the `.cabal` file describes the relevant version of a project and its dependencies:

Cabal file

As we told above, the `.cabal` file describes the relevant version of a project and its dependencies:

```
cabal-version: 1.12
name:           myFirstProject
version:        0.1.0.0
description:    Please see the README on GitHub at <https://github.com/githubuser/myFirstProject#readme>
homepage:       https://github.com/githubuser/myFirstProject#readme
bug-reports:    https://github.com/githubuser/myFirstProject/issues
author:         Author name here
maintainer:     example@example.com
copyright:      2020 Author name here
license:        BSD3
license-file:   LICENSE
build-type:     Simple
extra-source-files:
    README.md
    ChangeLog.md

source-repository head
  type: git
  location: https://github.com/githubuser/myFirstProject

library
  exposed-modules:
      Lib
  other-modules:
      Paths_myFirstProject
  hs-source-dirs:
      src
  build-depends:
```

`package.yaml`

The `package.yaml` is used to generate the `.cabal` file automatically:

package.yaml

The `package.yaml` is used to generate the `.cabal` file automatically:

```
name:                myFirstProject
version:             0.1.0.0
github:              "githubuser/myFirstProject"
license:             BSD3
author:              "Author name here"
maintainer:          "example@example.com"
copyright:           "2020 Author name here"

extra-source-files:
- README.md
- ChangeLog.md

description:         Please see the README on GitHub at <https://github.com/githubuser/myFirstProject#readme>

dependencies:
- base >= 4.7 && < 5

library:
  source-dirs: src

executables:
  myFirstProject-exe:
    main:                Main.hs
    source-dirs:         app
    ghc-options:
    - -threaded
    - -rtsopts
    - -with-rtsopts=-N
    dependencies:
    - myFirstProject
```

Building and running a project

The basic commands:

- `stack build`
- `stack run`
- `stack exec`
- `stack ghci`
- `stack clean`
- `stack test`

Building and running a project

The basic commands:

- `stack build`
- `stack run`
- `stack exec`
- `stack ghci`
- `stack clean`
- `stack test`

The roles of all these commands follow from their quite self-explanatory names.

Hackage

According to its description, ’Hackage is the Haskell community’s central package archive of open source software‘.

Hackage

According to its description, 'Hackage is the Haskell community's central package archive of open source software'.

- Webpage: `https://hackage.haskell.org`

Hackage

According to its description, 'Hackage is the Haskell community's central package archive of open source software'.

- Webpage: `https://hackage.haskell.org`
- Browsing packages, simplified package search, current uploads.

Hackage

According to its description, 'Hackage is the Haskell community's central package archive of open source software'.

- Webpage: `https://hackage.haskell.org`
- Browsing packages, simplified package search, current uploads.

type-natural: Type-level natural and proofs of their properties.

[bsd3, library, math] [Propose Tags]

Type-level natural numbers and proofs of their properties.

Version 0.6+ supports **GHC 8+ only**.

Use 0.5.* with ~ GHC 7.10.3.

Modules

[Index] [Quick Jump]

Data

Type

Data.Type.Natural

Data.Type.Natural.Builtin

Data.Type.Natural.Class

Data.Type.Natural.Class.Arithmetic

Data.Type.Natural.Class.Order

Data.Type.Ordinal

Data.Type.Ordinal.Builtin

Data.Type.Ordinal.Peano

Versions [faq]

0.0.1.0, 0.0.1.1, 0.0.2.0, 0.0.2.1, 0.0.3.0, 0.0.4.0, 0.0.5.0, 0.0.6.0, 0.1.0.0, 0.2.0.0, 0.2.1.0, 0.2.1.1, 0.2.1.2, 0.2.1.3, 0.2.1.4, 0.2.1.5, 0.2.2.0, 0.2.3.0, 0.2.3.1, 0.2.3.2, 0.3.0.0, 0.4.0.0, 0.4.1.0, 0.4.1.1, 0.4.2.0, 0.5.0.0, 0.6.0.0, 0.6.1.0, 0.6.1.1, 0.7.0.0, 0.7.1.0, 0.7.1.1, 0.7.1.2, 0.7.1.3, 0.7.1.4, 0.8.0.0, 0.8.0.1, 0.8.1.0, **0.8.2.0** (info)

Dependencies

base (==4.*), constraints (>=0.3), equational-reasoning (>=0.4.1.1), ghc-typelits-natnormalise (>=0.4), ghc-typelits-presburger (>=0.2.0.0), singletons (>=2.2 && <2.5), template-haskell (>=2.8) [details]

License

BSD-3-Clause

Copyright

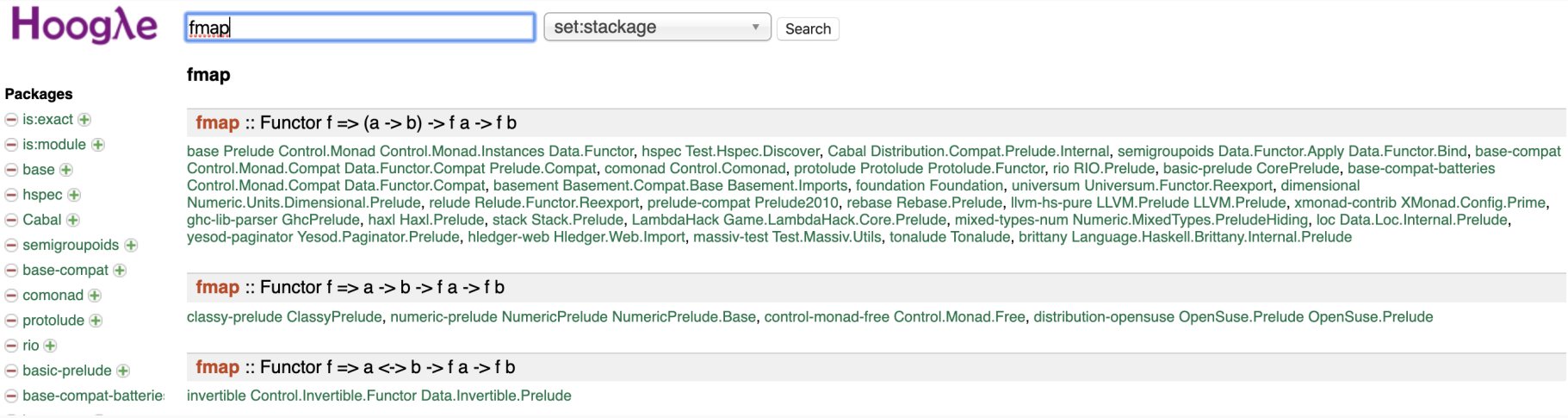
(C) Hiromi ISHII 2013-2014

Hoogle

Hoogle is a sort of Haskell search engine. Webpage: `https://hoogle.haskell.org`.

Hoogle

Hoogle is a sort of Haskell search engine. Webpage: `https://hoogle.haskell.org`.



Hackage Search

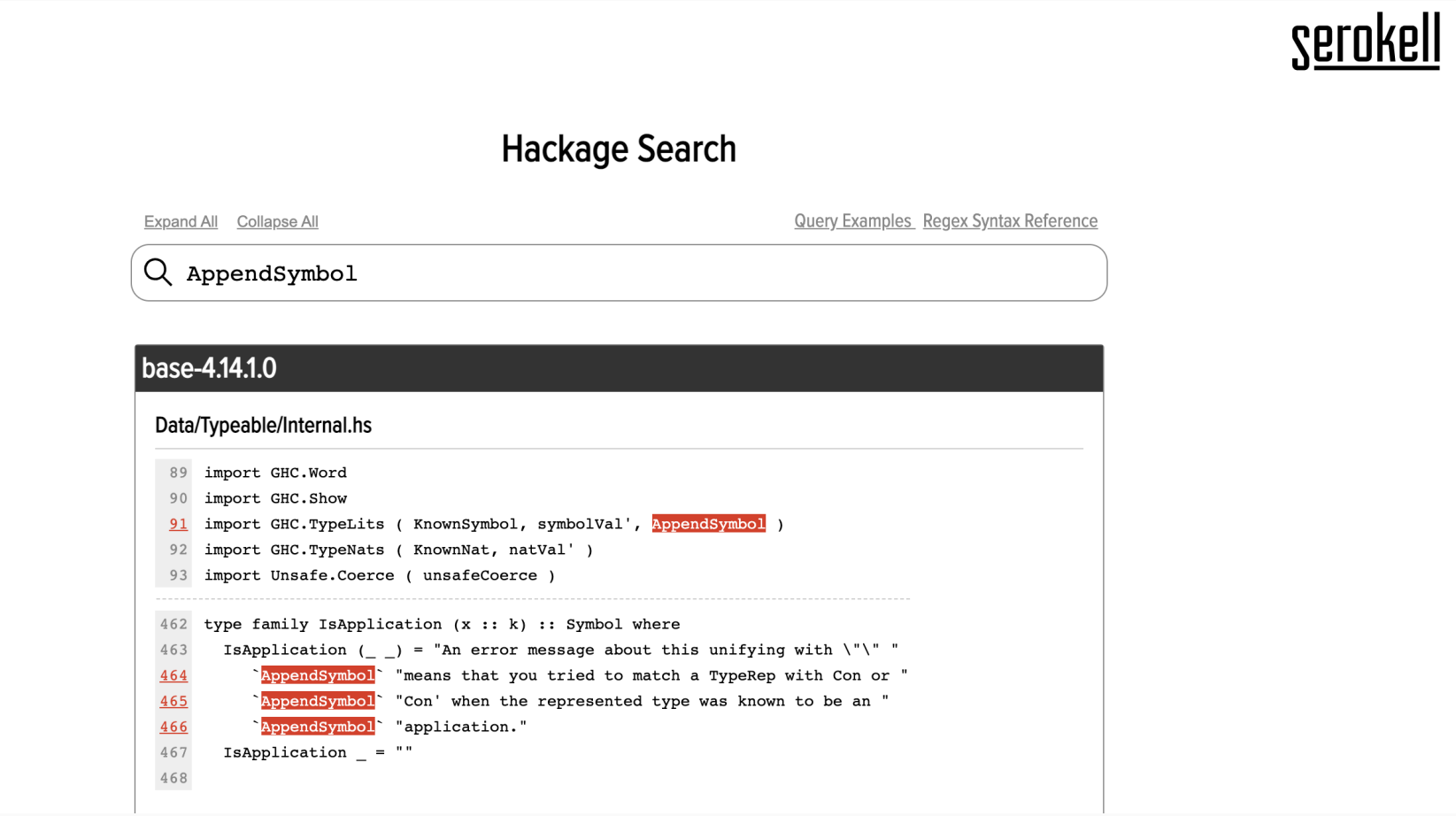
Hackage Search is a searching tool for Hackage based on regular expressions. This tool is by Vlad Zavialov, my GHC teammate from Serokell.

`https://hackage-search.serokell.io`.

Hackage Search

Hackage Search is a searching tool for Hackage based on regular expressions. This tool is by Vlad Zavialov, my GHC teammate from Serokell.

`https://hackage-search.serokell.io`.



Hackage Search

Hackage Search is a searching tool for Hackage based on regular expressions. This tool is made Vlad Zavialov, my GHC teammate from Serokell.

`https://hackage-search.serokell.io`.

Expand All Collapse All Query Examples Regex Syntax Reference

AppendSymbol

GHC/TypeLits.hs

```
47    , type (N.<=), type (N.<=?), type (N.+), type (N.*), type (N.^), type (N.-)
48    , type N.Div, type N.Mod, type N.Log2
49    , AppendSymbol
50    , N.CmpNat, CmpSymbol
51
```

```
147 --
148 -- @since 4.10.0.0
149 type family AppendSymbol (m ::Symbol) (n :: Symbol) :: Symbol
150
151 -- | A description of a custom type error.
```

changelog.md

```
262   * Add `plusForeignPtr` to `Foreign.ForeignPtr`.
263
264   * Add `type family AppendSymbol (m :: Symbol) (n :: Symbol) :: Symbol` to `GHC.TypeLits`
265     (#12162)
266
```

clash-ghc-1.2.5 | clash-lib-1.2.5 | clash-prelude-1.2.5 | dimensions-2.1.0.0 | easytensor-2.1.0.0 | facts-0.0.1.0
fcf-containers-0.5.0 | first-class-families-0.8.0.0 | generic-lens-2.0.0.0 | ghc-8.10.2 | ghc-lib-8.10.3.20201220 | harg-0.4.2.1
indigo-0.4 | lorentz-0.9.0 | morley-1.11.0 | mu-grpc-client-0.4.0.1 | mutable-0.2.2.0 | parquet-hs-0.1.0.0 | physics-0.1.2.1

Summary

We had a look at such topics as

1. General aspects of GHC and GHCi
2. The Haskell Platform installation
3. Dependency management using Stack and Cabal
4. In other words, the Haskell ecosystem in a nutshell

Summary

We had a look at such topics as

1. General aspects of GHC and GHCi
2. The Haskell Platform installation
3. Dependency management using Stack and Cabal
4. In other words, the Haskell ecosystem in a nutshell

On the next seminar, we will discuss:

1. The basic Haskell syntax
2. The underlying aspects of the Haskell type system
3. Functions and lambdas
4. Immutability and Laziness