Functional programming, Seminar No. 1

Daniel Rogozin
Institute for Information Transmission Problems, RAS
Serokell OÜ
Higher School of Economics
The Department of Computer Science

Basic intro

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

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 a primitive one. Informally, lambda-calculus is a formal system that describes 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 a primitive one. Informally, lambda-calculus is a formal system that describes 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 a primitive one. Informally, lambda-calculus is a formal system that describes 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 (due to Bertand Russell), 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 a primitive one. Informally, lambda-calculus is a formal system that describes 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 (due to Bertand Russell), 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 in Principia Mathematica (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 for 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 for imperative programming.

- We have no assignment in imperative languages. Variables are nullary constant functions rather than boxes.
- We have no states (in a usual sense)
- 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.

Types in programming languages are 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 (if you have M1, don't cry):

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

The Haskell Platform installation

There are several ways to install the Haskell platform on Mac (if you have M1, don't cry):

- 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 (if you have M1, don't cry):

- 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 (if you have M1, don't cry):

- 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.

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 has the following form:

  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, 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 closer.

```
cylindricalgebra@daniels-mbp ~ % ghci
GHCi, version 8.10.5: https://www.haskell.org/ghc/  :? for help
Loaded package environment from /Users/cylindricalgebra/.ghc/x86_64-dar
win-8.10.5/environments/default
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, and it still does.

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:

News

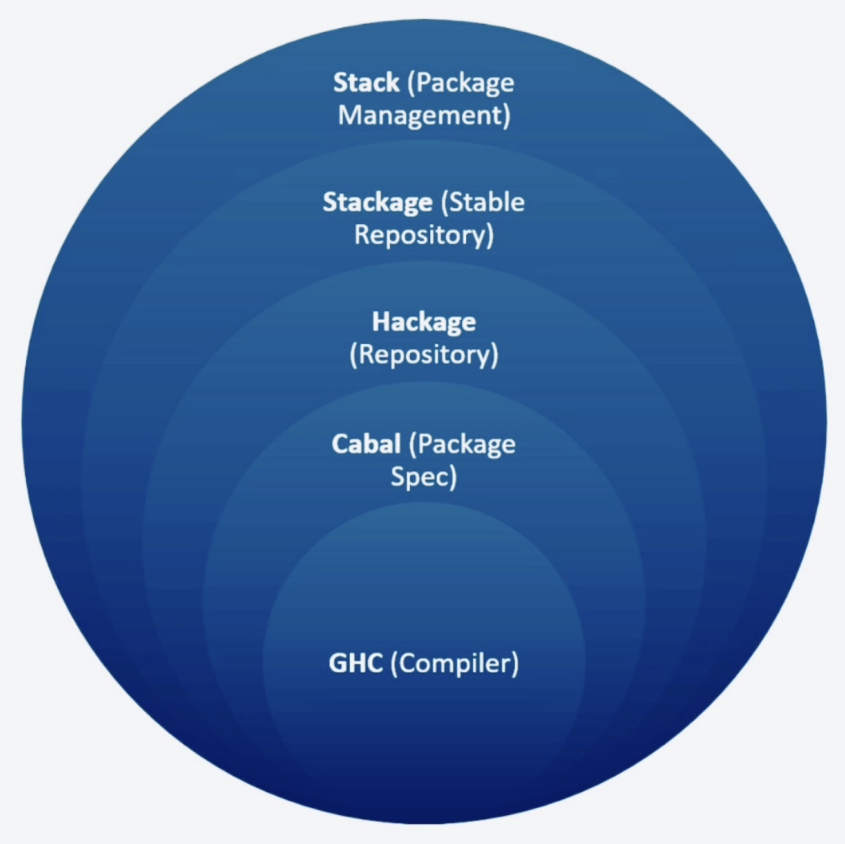
Stackage nightly snapshots to switch to GHC 9.0.1, 3 months ago

Latest releases per GHC version

- Stackage Nightly 2021-08-24 (ghc-9.0.1), today
- LTS 18.7 for ghc-8.10.6, published 4 days ago
- LTS 18.6 for ghc-8.10.4, published 4 days ago
- LTS 17.2 for ghc-8.10.3, published 7 months ago
- LTS 16.31 for ghc-8.8.4, published 7 months ago
- LTS 16.11 for ghc-8.8.3, published a year ago
- LTS 15.3 for ghc-8.8.2, published a year ago
- LTS 14.27 for ghc-8.6.5, published a year ago
- LTS 13.19 for ghc-8.6.4, published 2 years ago
- LTS 13.11 for ghc-8.6.3, published 2 years ago
- LTS 12.26 for ghc-8.4.4, published 3 years ago
- LTS 12.14 for ghc-8.4.3, published 3 years ago
- LTS 11.22 for ghc-8.2.2, published 3 years ago
- LTS 9.21 for ghc-8.0.2, published 4 years ago
- LTS 7.24 for ghc-8.0.1, published 4 years ago
- LTS 6.35 for ghc-7.10.3, published 4 years ago
- LTS 3.22 for ghc-7.10.2, published 6 years ago
- LTS 2.22 for ghc-7.8.4, published 6 years ago
- LTS 0.7 for ghc-7.8.3, published 7 years ago

Ecosystem encapsulation

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



Creating a Haskell project using Stack

- Name your project somehow 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 using Stack

- Name your project somehow and run the script `stack new <projectname>`
- You will see the following story after the command `tree .` in the project directory:

```
cylindricalgebra@daniels-mbp my-first-project % tree .
.
├── ChangeLog.md
├── LICENSE
├── README.md
├── Setup.hs
├── app
│   └── Main.hs
├── my-first-project.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 have a look the `stack.yaml` file:

stack.yaml

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

```yaml
resolver:
  url: https://raw.githubusercontent.com/commercialhaskell/stackage-snapshots/master/lts/18/7.yaml

# 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:
- .
# Dependency packages to be pulled from upstream that are not in the resolver.
# These entries can reference officially published versions as well as
# forks / in-progress versions pinned to a git hash. For example:
#
# extra-deps:
# - acme-missiles-0.3
# - git: https://github.com/commercialhaskell/stack.git
#   commit: e7b331f14bcffb8367cd58fbfc8b40ec7642100a
#
# extra-deps: []

# Override default flag values for local packages and extra-deps
# flags: {}
```

Cabal file

The `.cabal` file describes the relevant version of a project and its dependencies:

Cabal file

The `.cabal` file describes the relevant version of a project and its dependencies:

```
cabal-version: 1.12

-- This file has been generated from package.yaml by hpack version 0.34.4.
--
-- see: https://github.com/sol/hpack

name:           my-first-project
version:        0.1.0.0
description:    Please see the README on GitHub at <https://github.com/githubuser/my-first-project#readme>
homepage:       https://github.com/githubuser/my-first-project#readme
bug-reports:    https://github.com/githubuser/my-first-project/issues
author:         Author name here
maintainer:     example@example.com
copyright:      2021 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/my-first-project

library
```

`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:

```yaml
name:                my-first-project
version:             0.1.0.0
github:              "githubuser/my-first-project"
license:             BSD3
author:              "Author name here"
maintainer:          "example@example.com"
copyright:           "2021 Author name here"

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

# Metadata used when publishing your package
# synopsis:            Short description of your package
# category:            Web

# To avoid duplicated efforts in documentation and dealing with the
# complications of embedding Haddock markup inside cabal files, it is
# common to point users to the README.md file.
description:         Please see the README on GitHub at <https://github.com/githubuser/my-first-project#readm

dependencies:
- base >= 4.7 && < 5

library:
  source-dirs: src
```

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

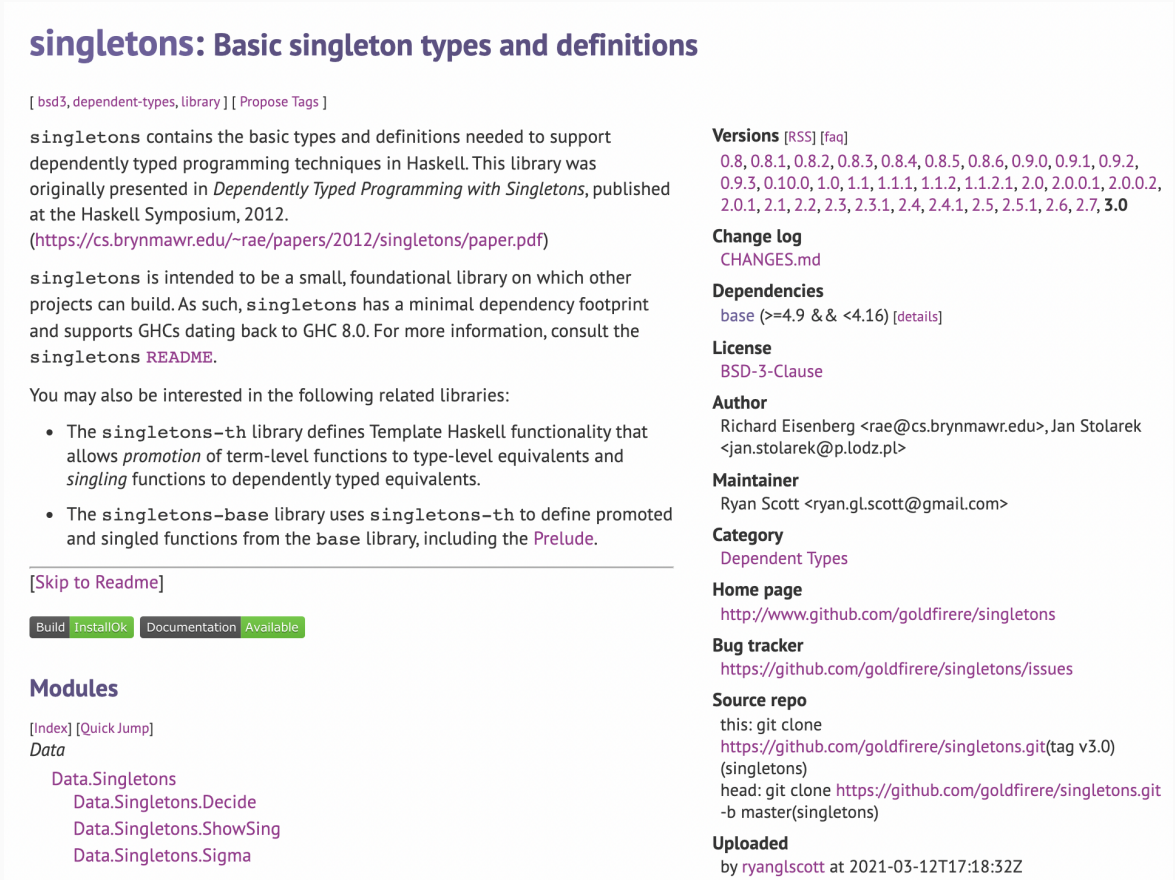
According to its description, 'Hackage is the Haskell community's central package archive of open source software'.

- 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.



singletons: Basic singleton types and definitions

[bsd3, dependent-types, library] [Propose Tags]

`singletons` contains the basic types and definitions needed to support dependently typed programming techniques in Haskell. This library was originally presented in *Dependently Typed Programming with Singletons*, published at the Haskell Symposium, 2012.
(https://cs.brynmawr.edu/~rae/papers/2012/singletons/paper.pdf)

`singletons` is intended to be a small, foundational library on which other projects can build. As such, `singletons` has a minimal dependency footprint and supports GHCs dating back to GHC 8.0. For more information, consult the `singletons` `README`.

You may also be interested in the following related libraries:

- The `singletons-th` library defines Template Haskell functionality that allows *promotion* of term-level functions to type-level equivalents and *singling* functions to dependently typed equivalents.
- The `singletons-base` library uses `singletons-th` to define promoted and singled functions from the `base` library, including the Prelude.

[Skip to Readme]

Build InstallOk Documentation Available

Modules

[Index] [Quick Jump]
Data
Data.Singletons
Data.Singletons.Decide
Data.Singletons.ShowSing
Data.Singletons.Sigma

Versions [RSS] [faq]
0.8, 0.8.1, 0.8.2, 0.8.3, 0.8.4, 0.8.5, 0.8.6, 0.9.0, 0.9.1, 0.9.2, 0.9.3, 0.10.0, 1.0, 1.1, 1.1.1, 1.1.2, 1.1.2.1, 2.0, 2.0.0.1, 2.0.0.2, 2.0.1, 2.1, 2.2, 2.3, 2.3.1, 2.4, 2.4.1, 2.5, 2.5.1, 2.6, 2.7, **3.0**

Change log
CHANGES.md

Dependencies
base (>=4.9 && <4.16) [details]

License
BSD-3-Clause

Author
Richard Eisenberg <rae@cs.brynmawr.edu>, Jan Stolarek <jan.stolarek@p.lodz.pl>

Maintainer
Ryan Scott <ryan.gl.scott@gmail.com>

Category
Dependent Types

Home page
http://www.github.com/goldfirere/singletons

Bug tracker
https://github.com/goldfirere/singletons/issues

Source repo
this: git clone https://github.com/goldfirere/singletons.git(tag v3.0) (singletons)
head: git clone https://github.com/goldfirere/singletons.git -b master(singletons)

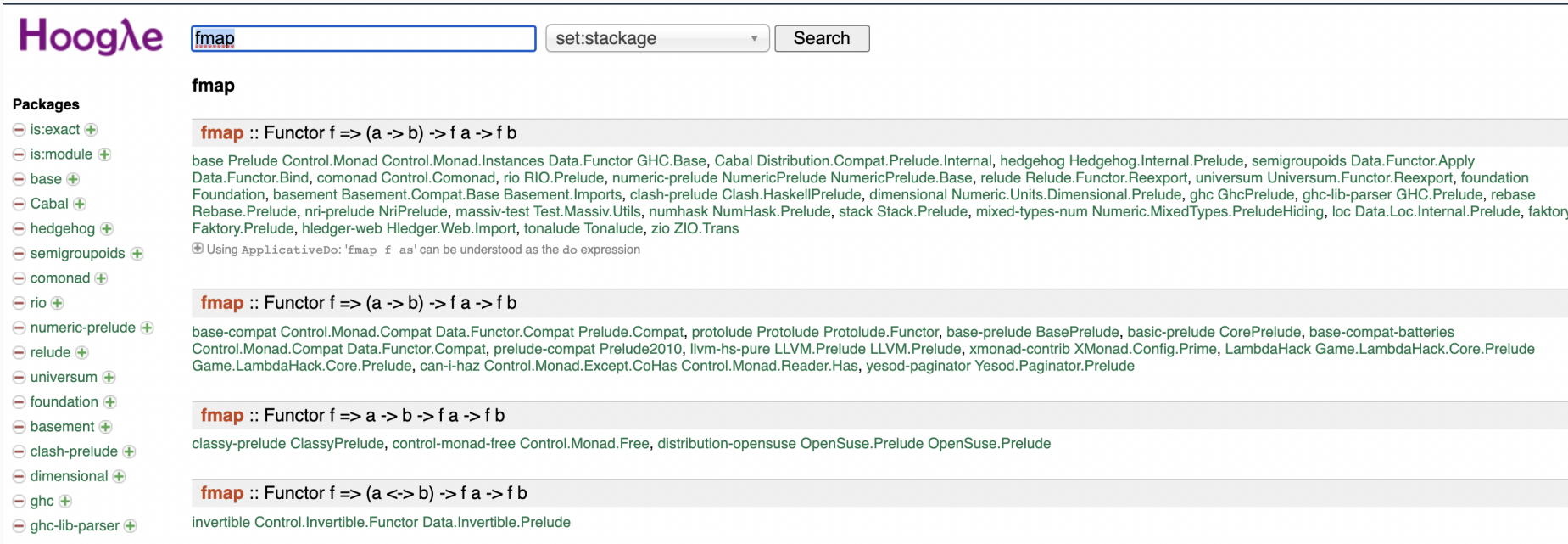
Uploaded
by ryanglscott at 2021-03-12T17:18:32Z

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

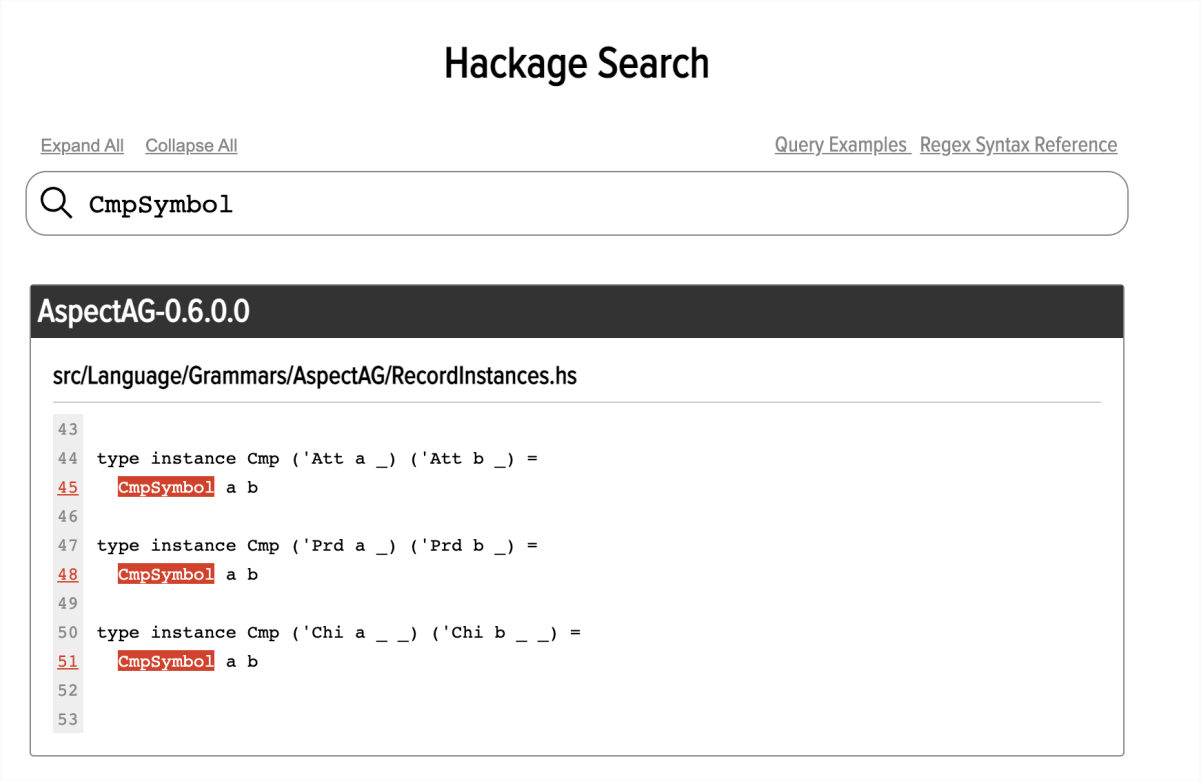
Hackage Search is a searching tool for Hackage based on regular expressions. This tool is by Vladislav Zavialov, my GHC teammate from Serokell.

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

Hackage Search

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

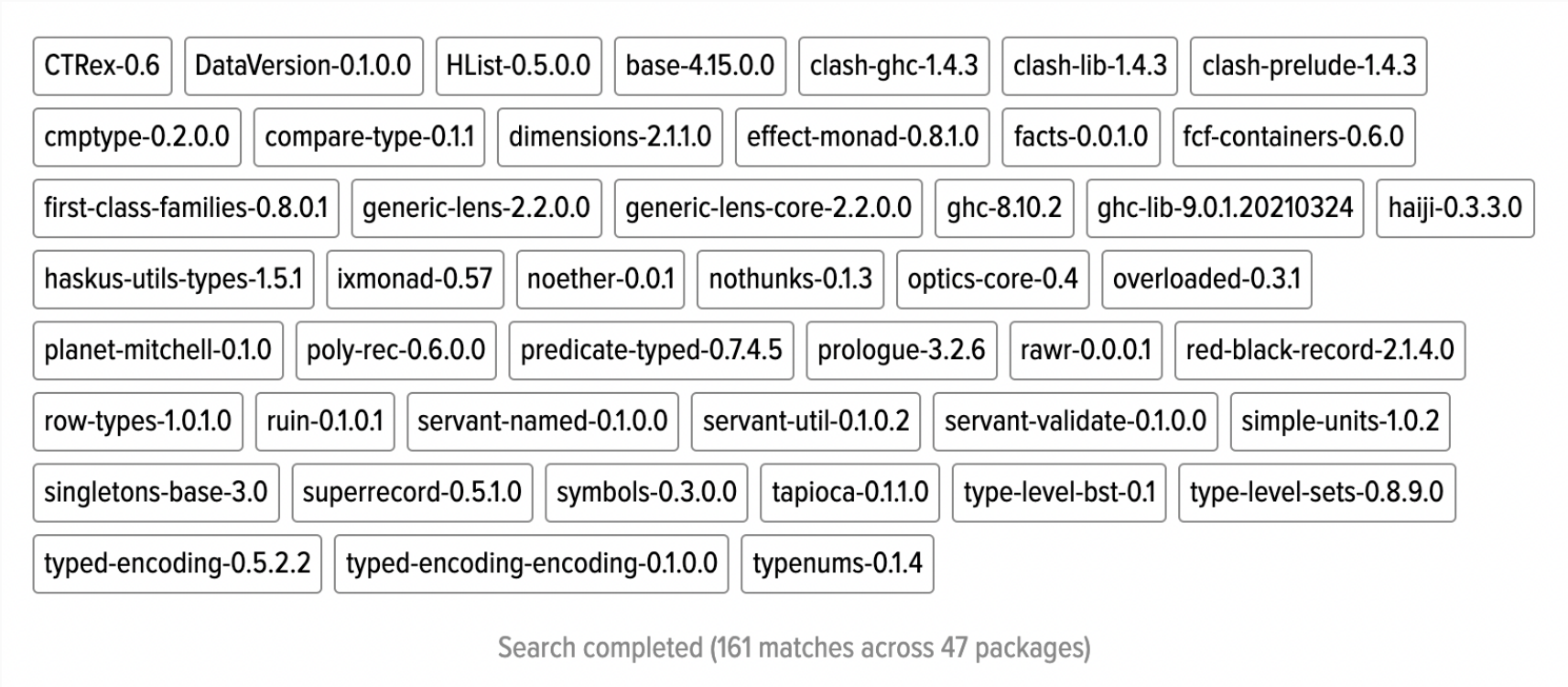
`https://hackage-search.serokell.io`.



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`.



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