A quick overview of the Haskell tooling

Julien Dehos

Fosdem 2023

Who am I?

- assistant professor in Computer Science at ULCO, France
- using Haskell since 2015 (for teaching FP + small projects)

50 shades of Haskell tooling

- compiler: GHC
- build tools: cabal, stack
- editors (+ plugins): vscode, vim, emacs
- LSP implementation: HLS
- setup tools: ghcup, nix

→ in this talk: cabal, vscode + haskell.haskell

Hackage

- community package archive
- https://hackage.haskell.org

Hackage :: [Package]

Search · Browse · What's new · Upload · User accounts

Hackage: The Haskell Package Repository

Hackage is the Haskell community's central package archive of open source software. Hackage has been online since January 2007 and is constantly growing. You can publish libraries and programs or download and install packages with tools like cabal-install (or via your distro's package manager).

Package Downloads

- Browse packages
- Search packages
- Recent uploads

Supplementary views:

- Packages by category
- Tags
- Deprecated packages
- Candidate packages

- online documentation:

Hackage :: [Package]

Search · Browse · What's new · Upload · User accounts

vector: Efficient Arrays

[bsd3, data, data-structures, library] [Propose Tags]

An efficient implementation of `Int`-indexed arrays (both mutable and immutable), with a powerful loop optimisation framework .

It is structured as follows:

Data.Vector
: Boxed vectors of arbitrary types.

Data.Vector.Unboxed
: Unboxed vectors with an adaptive representation based on data type families.

Data.Vector.Storable
: Unboxed vectors of `Storable` types.

Data.Vector.Primitive
: Unboxed vectors of primitive types as defined by the `primitive` package. Data.Vector.Unboxed is more flexible at no performance cost.

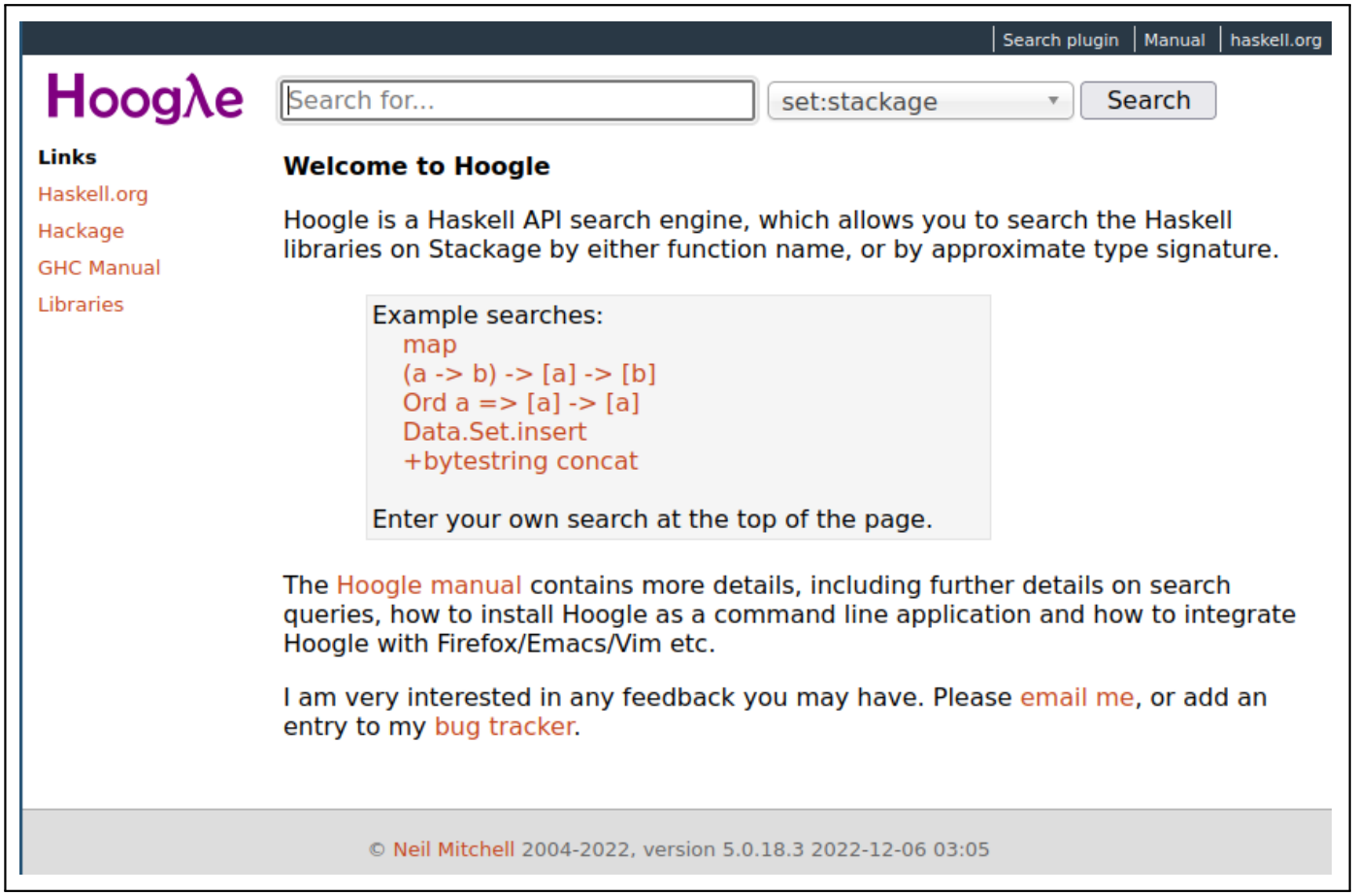
- online source code:

```haskell
instance Monad Vector where
  {-# INLINE return #-}
  return = Applicative.pure

  {-# INLINE (>>=) #-}
  (>>=) = flip concatMap

#if !(MIN_VERSION
  {-# INLINE fail
  fail = Fail.fai
#endif

-- | @since 0.12.1.0
instance Fail.MonadFail Vector where
  {-# INLINE fail #-}
  fail _ = empty

instance MonadPlus Vector where
  {-# INLINE mzero #-}
  mzero = empty

  {-# INLINE mplus #-}
  mplus = (++)
```

```haskell
(a -> Vector b) -> Vector a -> Vector b
forall a b. (a -> Vector b) -> Vector a -> Vector b
```

Hoogle

- "Haskell google"
- https://hoogle.haskell.org



- search by function name or by type signature:

Hoogλe

(a -> b) -> [a] -> [b]

set:stackage

Search

Packages

- is:exact
- base
- ghc
- haskell-gi-base
- ihaskell
- ghc-lib-parser
- rebase
- xmonad-contrib
- faktory
- freckle-app
- hledger-web
- base-prelude
- rio
- numeric-prelude
- relude
- dimensional
- mixed-types-num

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

map :: (a -> b) -> [a] -> [b]

base Prelude Data.List GHC.Base GHC.List GHC.OldList, ghc GHC.Prelude, haskell-gi-base Data.GI.Base.ShortPrelude, ihaskell IHaskellPrelude, ghc-lib-parser GHC.Prelude, rebase Rebase.Prelude, xmonad-contrib XMonad.Config.Prime, faktory Faktory.Prelude, freckle-app Freckle.App.Prelude Freckle.App.Test, hledger-web Hledger.Web.Import

map f xs is the list obtained by applying f to each element of xs, i.e.,

strictMap :: (a -> b) -> [a] -> [b]

ghc GHC.Utils.Misc, ghc-lib-parser GHC.Utils.Misc

map :: (a -> b) -> [a] -> [b]

base-prelude BasePrelude, rio RIO.List RIO.Prelude, numeric-prelude NumericPrelude NumericPrelude.Base, relude Relude.List.Reexport, dimensional Numeric.Units.Dimensional.Prelude, mixed-types-num Numeric.MixedTypes.PreludeHiding, Cabal-syntax Distribution.Compat.Prelude, numhask NumHask.Prelude, stack Stack.Prelude, LambdaHack Game.LambdaHack.Core.Prelude Game.LambdaHack.Core.Prelude, yesod-paginator Yesod.Paginator.Prelude, distribution-opensuse OpenSuse.Prelude, cabal-install-solver Distribution.Solver.Compat.Prelude

map f xs is the list obtained by applying f to each element of xs, i.e.,

Cabal

- system for building and packaging Haskell libraries and programs
- write a cabal file:

myproject.cabal

```
cabal-version:      3.4
name:               myproject
version:            0.1.0.0
description:
    This is myproject.

common shared-properties
    default-language: Haskell2010
    ghc-options:      -Wall

library
    import:           shared-properties
    build-depends:    base
    hs-source-dirs:   src
    exposed-modules:  Tree

executable myproject
    import:           shared-properties
    build-depends:    base, myproject
    hs-source-dirs:   app
    main-is:          Main.hs
```

- run the cabal tool:

```
[nix-shell:~/code/myproject]$ cabal build
Build profile: -w ghc-9.0.2 -O1
In order, the following will be built (use -v for more details):
 - myproject-0.1.0.0 (exe:myproject) (file app/Main.hs changed)
Preprocessing executable 'myproject' for myproject-0.1.0.0..
Building executable 'myproject' for myproject-0.1.0.0..
[1 of 1] Compiling Main             ( app/Main.hs, /home/julien/code/mypro
ject/dist-newstyle/build/x86_64-linux/ghc-9.0.2/myproject-0.1.0.0/x/myproj
ect/build/myproject/myproject-tmp/Main.o )
Linking /home/julien/code/myproject/dist-newstyle/build/x86_64-linux/ghc-9
.0.2/myproject-0.1.0.0/x/myproject/build/myproject/myproject ...

[nix-shell:~/code/myproject]$ cabal run myproject data/example1.txt
Up to date
list: [1,2,2,3,3,8,23,34,43,52,123,432,432,541,893]
sum: 2592
length: 15
```

REPL

- Read-Eval-Print Loop

```
[nix-shell:~/code/myproject]$ cabal repl
Build profile: -w ghc-9.0.2 -O1
In order, the following will be built (use -v for more details):
 - myproject-0.1.0.0 (lib) (ephemeral targets)
Preprocessing library for myproject-0.1.0.0..
GHCi, version 9.0.2: https://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Tree             ( src/Tree.hs, interpreted )
Ok, one module loaded.
ghci>
ghci> import Data.Foldable
ghci>
ghci> :info Tree
type Tree :: * -> *
data Tree v = Leaf | Node (Tree v) v (Tree v)
        -- Defined at src/Tree.hs:10:1
instance [safe] Eq v => Eq (Tree v) -- Defined at src/Tree.hs:13:15
instance [safe] Foldable Tree -- Defined at src/Tree.hs:13:19
instance [safe] Show v => Show (Tree v)
  -- Defined at src/Tree.hs:13:29
ghci>
ghci> mytree = fromList [2, 1, 4, 3]
ghci>
ghci> toList mytree  :: [Int]
[1,2,3,4]
ghci>
ghci>
```

Visual Studio Code (+ HLS + haskell.haskell)

- file edition, syntax highlighting:

Tree.hs - myproject - Visual Studio Code

Fichier Edition Sélection Affichage Atteindre Exécuter Terminal Aide

EXPLORATEUR

MYPROJECT
- .vscode
- app
  - Main.hs
- data
  - example1.txt
- dist-newstyle
- src
  - Tree.hs
- test
  - Spec.hs
  - TreeSpec.hs
- default.nix
- myproject.cabal
- README.md

STRUCTURE

CHRONOLOGIE

Main.hs | Tree.hs

src > Tree.hs > Tree

```haskell
{-# Language DeriveFoldable #-}

-- | This is the Tree module
module Tree where

import Data.List ( foldl' )

-- | A polymorphic binary Tree
data Tree v
    = Leaf
    | Node (Tree v) v (Tree v)
    deriving (Eq, Foldable, Show)

-- | Build a Tree from a List
--
Refresh...
-- >>> fromList [2,1,3] :: Tree Int
-- Node (Node Leaf 1 Leaf) 2 (Node Leaf 3 Leaf)
fromList :: Ord a => [a] -> Tree a
fromList = foldl' insert1 Leaf
```

Environment: default.nix ⊗ 0 ⚠ 0 L 13, col 1 Espaces : 4 UTF-8 LF Haskell

- code navigation/documentation:

Main.hs - myproject - Visual Studio Code

Fichier Edition Sélection Affichage Atteindre Exécuter Terminal Aide

EXPLORATEUR

MYPROJECT
- .vscode
- app
  - Main.hs
- data
  - example1.txt
- dist-newstyle
- src
  - Tree.hs
- test
  - Spec.hs
  - TreeSpec.hs
- default.nix
- myproject.cabal
- README.md

STRUCTURE

CHRONOLOGIE

Main.hs | Tree.hs

app > Main.hs > ...

```
1  import Data.Foldable ( Foldable(toList) )
2  import System.Environment ( getArgs )
3
   import Tree ( Tree, fromList )
4  import Tree
5
6  main :: IO ()
7  main = do
8      args <- getArgs
9      case args of
10
11         [filename] ->
12             contents
13             let xs = fromList (read <$> words contents) :: Tree Int
14             putStrLn $ "list: " ++ show (toList xs)
15             putStrLn $ "sum: " ++ show (sum xs)
16             putStrLn $ "length: " ++ show (length xs)
17
18         _ -> putStrLn "usage: <filename>"
19
```

```
$dOrd :: Ord Int
fromList :: forall a. Ord a => [a] -> Tree a
```

Defined at /home/julien/code/myproject/src/Tree.hs:20:1

Build a Tree from a List

```
>>> fromList [2,1,3] :: Tree Int
Node (Node Leaf 1 Leaf) 2 (Node Leaf 3 Leaf)
```

```
_ :: [Int] -> Tree Int
_ :: forall a. Ord a => [a] -> Tree a
```

Environment: default.nix ⊗ 0 ⚠ 0 L 20, col 1 Espaces : 4 UTF-8 LF Haskell

- code completion:

Tree.hs - myproject - Visual Studio Code

Fichier Edition Sélection Affichage Atteindre Exécuter Terminal Aide

EXPLORATEUR

MYPROJECT
- .vscode
- app
  - Main.hs
- data
  - example1.txt
- dist-newstyle
- src
  - Tree.hs 1
- test
  - Spec.hs
  - TreeSpec.hs
- default.nix
- myproject.cabal
- README.md

STRUCTURE

CHRONOLOGIE

Main.hs | Tree.hs 1

src > Tree.hs > Tree

```
36          (Node
37          2
38          (Node
39
40  -- | The same
41  --
    Refresh...
42  -- >>> fromLi
43  -- Node (Node
44  tree2 :: Tree
45  tree2 = fromL
46
47  mysum :: Num v => Tree v -> v
48  mysum Leaf = 0
49  mysum (Node left val right) = mysum left + val + mysum right
50
    Refresh...
51  -- >>> mysum tree2
52  -- 6
53
```

fromList :: Ord a => [a] -> Tree a
fromLabel
fromLeft
fromListN
fromCallSiteList
fromCallSiteList
fromCallSiteList
fromBool
fromBool
fromBool
fromBool
fromBool

Environment: default.nix ⊗ 1 ⚠ 0 L 45, col 14 Espaces : 4 UTF-8 LF Haskell

- compilation, refactoring:

Main.hs - myproject - Visual Studio Code

Fichier Edition Sélection Affichage Atteindre Exécuter Terminal Aide

EXPLORATEUR

MYPROJECT
- .vscode
- app
  - Main.hs
- data
  - example1.txt
- dist-newstyle
- src
  - Tree.hs 6
- test
  - Spec.hs
  - TreeSpec.hs
- default.nix
- myproject.cabal
- README.md

STRUCTURE

CHRONOLOGIE

Main.hs

app > Main.hs > main

```haskell
import Tree

main :: IO ()
main = do
    args <- getArgs
    case args of

        [filename] -> do
            contents <- readFi
            let xs = fromList (fmap read $ words contents) :: Tree Ir
            putStrLn $ "list: " ++ show (toList xs)
            putStrLn $ "sum: " ++ show (sum xs)
            putStrLn $ "length: " ++ show (length xs)

        _ -> putStrLn "usage: <filename>"
```

Use <$>
Found:
  fmap read $ words contents
Why not:
  read <$> words contents
hlint(refact:Use <$>)

Voir le problème Correction rapide... (Ctrl+.)

Environment: default.nix 4 2 1 L 13, col 44 Espaces : 4 UTF-8 LF Haskell

- holes:

Main.hs - myproject - Visual Studio Code

Fichier Edition Sélection Affichage Atteindre Exécuter Terminal Aide

EXPLORATEUR

MYPROJECT
- .vscode
- app
  - Main.hs 1
- data
  - example1.txt
- dist-newstyle
- src
  - Tree.hs
- test
  - Spec.hs
  - TreeSpec.hs
- default.nix
- myproject.cabal
- README.md

STRUCTURE

CHRONOLOGIE

Main.hs 1 × Tree.hs

app > Main.hs > main

```haskell
import Data.Foldable (
import System.Environme

import Tree ( Tree, fromList )
import Tree

main :: IO ()
main = do
    args <- getArgs
    case args of

        [filename] -> c
            contents <-
            let xs = fromList (_ <$> words contents) :: Tree Int
            putStrLn $ "list: " ++ show (toList xs)
            putStrLn $ "sum: " ++ show (sum xs)
            putStrLn $ "length: " ++ show (length xs)

        _ -> putStrLn "usage: <filename>"
```

```haskell
_ :: String -> Int
```

```
• Found hole: _ :: String -> Int
• In the first argument of '(<$>)', namely '_'
  In the first argument of 'fromList', namely
    '(_ <$> words contents)'
  In the expression: fromList (_ <$> words contents)
:: Tree Int
• Relevant bindings include
    xs :: Tree Int
      (bound at
/home/julien/code/myproject/app/Main.hs:13:17)
    contents :: String
```

Environment: default.nix ⊗ 1 ⚠ 0 L 18, col 40 Espaces : 4 UTF-8 LF Haskell

- inline REPL:

Main.hs - myproject - Visual Studio Code

Fichier Edition Sélection Affichage Atteindre Exécuter Terminal Aide

EXPLORATEUR

MYPROJECT
- .vscode
- app
  - Main.hs
- data
  - example1.txt
- dist-newstyle
- src
  - Tree.hs
- test
  - Spec.hs
  - TreeSpec.hs
- default.nix
- myproject.cabal
- README.md

STRUCTURE

CHRONOLOGIE

Main.hs ● Tree.hs

app > Main.hs > ...

```
1   import Data.Foldable ( Foldable(toList) )
2   import System.Environment ( getArgs )
3
    import Tree ( fromList, Tree )
4   import Tree
5
    Refresh...
6   -- >>> tree1
7   -- >>> toList tree1
8   -- Node (Node Leaf 1 Leaf) 2 (Node Leaf 3 Leaf)
9   -- [1,2,3]
10
11  main :: IO ()
12  main = do
13      args <- getArgs
14      case args of
15
16          [filename] -> do
17              contents <- readFile filename
18              let xs = fromList (read <$> words contents) :: Tree Int
19              putStrLn $ "list: " ++ show (toList xs)
```

Environment: default.nix ⊗ 0 ⚠ 0 L 6, col 13 Espaces : 4 UTF-8 LF Haskell

Haddock

- document the code:

src > Tree.hs > Tree

```haskell
{-# Language DeriveFoldable #-}

-- | This is the Tree module
module Tree where

import Data.List ( foldl' )

-- | A polymorphic binary Tree
data Tree v
    = Leaf
    | Node (Tree v) v (Tree v)
    deriving (Eq, Foldable, Show)

-- | Build a Tree from a List
--
Refresh...
-- >>> fromList [2,1,3] :: Tree Int
-- Node (Node Leaf 1 Leaf) 2 (Node Leaf 3 Leaf)
fromList :: Ord a => [a] -> Tree a
fromList = foldl' insert1 Leaf
  where
```

- generate the documentation:

```
[nix-shell:~/code/myproject]$ cabal haddock
Build profile: -w ghc-9.0.2 -O1
In order, the following will be built (use -v for more details):
 - myproject-0.1.0.0 (lib) (file src/Tree.hs changed)
./myproject.cabal has been changed. Re-configuring with most recently used
options. If this fails, please run configure manually.
Configuring library for myproject-0.1.0.0..
Preprocessing library for myproject-0.1.0.0..
Running Haddock on library for myproject-0.1.0.0..
  83% (  5 /  6) in 'Tree'
  Missing documentation for:
    mysum (src/Tree.hs:47)
Documentation created:
/home/julien/code/myproject/dist-newstyle/build/x86_64-linux/ghc-9.0.2/myproject
-0.1.0.0/doc/html/myproject/index.html
```

- result:

Documentation

Synopsis

```
data Tree v                                                    #
```

A polymorphic binary Tree

Constructors

```
Leaf
Node (Tree v) v (Tree v)
```

▽ **Instances**

```
▷ Foldable Tree                #
▷ Show v => Show (Tree v)      #
▷ Eq v => Eq (Tree v)          #
```

```
fromList :: Ord a => [a] -> Tree a                             #
```

Build a Tree from a List

```
>>> fromList [2,1,3] :: Tree Int
Node (Node Leaf 1 Leaf) 2 (Node Leaf 3 Leaf)
```

Tests

- unit tests, with HSpec:

test > TreeSpec.hs > TreeSpec > spec

```haskell
main :: IO ()
main = hspec spec

spec :: Spec
spec = do

    describe "fromList" $ do
        it "[1,2,3]" $ fromList [1,2,3::Int]
            `shouldBe` Node Leaf 1 (Node Leaf 2 (Node Leaf 3 Leaf))
        it "[2,1,3]" $ fromList [2,1,3::Int]
            `shouldBe` Node (Node Leaf 1 Leaf) 2 (Node Leaf 3 Leaf)

    describe "toList . fromList" $ do
        it "[1,2,3]" $ toList (fromList [1,2,3]) `shouldBe` [1,2,3]
        it "[2,1,3]" $ toList (fromList [2,1,3]) `shouldBe` [1,2,3]
```

- results:

```
[nix-shell:~/code/myproject]$ cabal test --test-show-details=always
Build profile: -w ghc-9.0.2 -O1
In order, the following will be built (use -v for more details):
 - myproject-0.1.0.0 (test:spec) (first run)
Preprocessing test suite 'spec' for myproject-0.1.0.0..
Building test suite 'spec' for myproject-0.1.0.0..
Running 1 test suites...
Test suite spec: RUNNING...

Tree
  fromList
    [1,2,3]
    [2,1,3]
  toList . fromList
    [1,2,3]
    [2,1,3]

Finished in 0.0001 seconds
4 examples, 0 failures

Test suite spec: PASS
Test suite logged to:
/home/julien/code/myproject/dist-newstyle/build/x86_64-linux/ghc-9.0.2/myproject-0.1.0.
0/t/spec/test/myproject-0.1.0.0-spec.log
1 of 1 test suites (1 of 1 test cases) passed.
```

- test properties, with QuickCheck:

test > TreeSpec.hs > TreeSpec > spec

```haskell
    let
        prop_sort :: [Int] -> Bool
        prop_sort = isSorted . toList . fromList

    describe "quickCheck" $ do
        it "prop_sort" $ property prop_sort
```

- result:

```
Test suite spec: RUNNING...

Tree
  fromList
    [1,2,3]
    [2,1,3]
  toList . fromList
    [1,2,3]
    [2,1,3]
  quickCheck
    prop_sort
      +++ OK, passed 100 tests.
```

Conclusion

- Haskell has some nice tools for many years (cabal, repl, QuickCheck. . .)
- and intuitive tools since recently (vscode plugins, HLS. . .)
- through a quite easy setup : ghcup + vscode + haskell.haskell

References

- slides & code:
  https://gitlab.com/juliendehos/talk-2023-fosdem
- 2022 State of Haskell Survey Results:
  https://taylor.fausak.me/2022/11/18/haskell-survey-results
- The Varieties of the Haskelling Experience:
  https://www.tweag.io/blog/2021-11-25-varieties-of-haskelling-experience

Thank you! Questions/discussion?