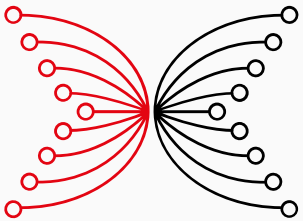
Packaging and tools

Haskell and Cryptocurrencies

Dr. Lars Brünjes, IOHK
Alejandro Garcia, IOHK
Dr. Andres Löh, Well-Typed LLP

2020-07-27



INPUT | OUTPUT

Goals

- Modules and packages.
- Cabal the library and Hackage.
- cabal(-install) the tool.
- Interface documentation, Haddock, Hoogle.

Modules

Modules

- Haskell programs are organized into modules.
- Module names start with capital letters and are hierarchical, i.e., consist of possibly multiple components, such as `Data.List` or `Control.Concurrent.Async`.
- One module per file, named as the last component of the module. Initial components of module names are mapped to directories, so that the compiler can find them.
- Every full Haskell program must have a module called `Main` which is the entry module, and within it, a binding for `main` must be made (more about that in the next lecture).

Module header

Example:

```
module MyModule where

import Control.Applicative
import Data.List
import Data.Maybe
```

- If the `module` declaration is missing, the module name is `Main`.
- The module called `Prelude` is implicitly imported if there is no explicit `import` statement for it.
- All `import` declarations must be at the top of the module after the `module` declaration.

Name management

Example:

```
module Tree (Tree (..), elem, sort) where
import Prelude hiding (elem)
```

- An export list restricts the names that are exported.
- With `Tree (..)`, a datatype and all constructors are exported.
- Import lists restrict the names that are imported.
- Some names can also be hidden via `hiding`.
- Without export / import lists, everything[1] is exported / imported.

[1] In the case of exports, only locally defined entities are exported by default

We still need to define what we export

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
```

Constructors:

```
Leaf :: a -> Tree a
Node :: Tree a -> Tree a -> Tree a
```

Functions on trees:

```
fun :: Tree a -> ...
fun (Leaf x) = ...
fun (Node l r) = ...fun l ... fun r ...
```

Recursion in types and functions are connected!

Tree element check

```haskell
elem :: Eq a => a -> Tree a -> Bool
a `elem` Leaf a'  = a == a'
a `elem` Node l r = a `elem` l || a `elem` r
```

Tree merge sort

```haskell
sort :: Ord a => Tree a -> [a]
sort (Leaf x)   = [x]
sort (Node l r) = merge (sort l) (sort r)
```

```haskell
merge :: Ord a => [a] -> [a] -> [a]
merge []       ys = ys
merge (x : xs) [] = x : xs
merge (x : xs) (y : ys)
  | x <= y     = x : merge      xs (y : ys)
  | otherwise = y : merge (x : xs)     ys
```

More that we will cover when needed

- Qualified inputs and qualified names.
- Renaming modules when importing.
- Re-exporting imported modules.
- Exporting / importing of type classes and instances.

Packages

Packages

- Modules are organized into packages.
- Modules are part of the language, packages are part of the compiler infrastructure.
- We need to tell the compiler (GHC) which packages we use in order to make their modules accessible in our program.
- The compiler comes with a package `base` which is accessible by default.
- The `Prelude` is a part of `base`, as are many other important modules.

Hackage

- Hackage is the main package repository in the Haskell world.
- Lots of useful open-source packages and applications on all aspects of programming.
- A mix of widely used, actively maintained, and esoteric or hobbyist packages.
- Hackage can be browsed, and contains lots of info and documentation on packages.

Cabal

Cabal (Common Architecture for Building Applications and Libraries) is

- the Haskell package format, and
- the name of the library supporting the management and building of packages.

Standard directory layout of a package

```
/mypackage
/mypackage/mypackage.cabal
/mypackage/cabal.project
/mypackage/Setup.hs
/mypackage/src
...
```

Demo

Package management

Demo