Haskell Typing Specifiers

Principles of Programming Languages

Colorado School of Mines

https://lambda.mines.edu

Learning Group Activity

Spend 10 minutes comparing your code from the LGA with your learning groups' code. How did your implementations differ? Share anything cool you did too!

Haskell Type Specifiers

Haskell uses inferred static typing. However, you may specify the types of a function if you wish.

Haskell Type Specifiers

Haskell uses inferred static typing. However, you may specify the types of a function if you wish.

To declare the type of a function in Haskell, use the following notation:

```
double :: Int -> Int
double x = x * 2
```

This declaration reads

`double` is a function which takes an `Int` and returns an `Int`.

Review: Currying

- Haskell takes advantage of **currying** to support functions with multiple arguments. That is, functions take a single argument and return a function ready to take the next argument.
- We call the function ready to take the next argument a **partially applied function**.

Multiple Arguments

Declaring types of a function gets a little more tricky once we involve currying:

```
gcd :: Int -> Int -> Int
gcd a b = ...
```

What exactly does this read?

`gcd` is a function which takes an `Int`, and returns a function that takes an `Int` and returns an `Int`.

Lists in Type Specifiers

To specify a list of some type, surround the type in square brackets. For example:

```
frobnicate :: Int -> [Char] -> [Char]
frobnicate n st = ...
```

Lists in Type Specifiers

To specify a list of some type, surround the type in square brackets. For example:

```
frobnicate :: Int -> [Char] -> [Char]
frobnicate n st = ...
```

However, this might be more cleanly written as:

```
frobnicate :: Int -> String -> String
frobnicate n st = ...
```

as `String` is just a synonym for `[Char]`.

Type Variables

Haskell has a cool little trick: **type variables**. Consider this type specifier for the head function:

```
head :: [a] -> a
```

Type Variables

Haskell has a cool little trick: **type variables**. Consider this type specifier for the head function:

```
head :: [a] -> a
```

This reads

head takes a list of **some type** a, and returns something of type a.

This makes sense. head can be preformed on lists of any type, and we can guarantee it will give us something of the type the list is of.

Typeclasses

Suppose that we wanted to specify restrictions on what our type variables can be. Haskell features **typeclasses** for this. For example, what if we wanted to declare the type of a `sort` function:

```
sort :: (Ord a) => [a] -> [a]
```

This reads

`sort` takes a list of type a, and returns a list of type a, so as long as the type a is something that can be ordered.

More Typeclasses

- `Num a`: a is numeric
- `Eq a`: a can be tested for equality
- `Integral a`: a is an integer (either bounded or unbounded)
- More examples in LYH.

Querying GHC for Inferred Type

Wondering what GHC guessed? Use `:t` in GHCi to check the type of a function:

```
GHCi> :t fst
fst :: (a, b) -> a
```

Quiz Study Time

1. As a learning group, create a quiz that you think I could have potentially written for you today.
2. When done, trade quizzes with another learning group and complete their quiz.

I am also happy to answer any questions or go over old slides during this time. Plan is to start the real quiz in about 15 to 20 minutes.