CS 350 2024-25 Sem I Lecture 7

Satyadev Nandakumar

August 23, 2024

Outline

Outine

"First-class" values

If a user-defined type has all features of a default type, then we must be able to

1. assign value of that type to variables
2. pass values of that type as arguments to functions
3. return values of that type from functions
4. store values of that type in other data structures
5. do pattern-matching on values of that type

We can do all these!

Introduction

Haskell provides facilities for users to define new types, and for defining "classes of types"

Ways to define new types in Haskell

1. using `type`
2. using `data` [most important]
3. using `newtype`

Classes

classes are "groups" of similar types. e.g.

1. `Eq` is a class of types that supports equality
2. `Ord` is a class of types that supports linear order (<=)
3. `Num` is a class of numerical types

Types are instances of classes. e.g.

1. `Bool` and `Int` are instances of `Eq`
2. They are also instances of `Ord`. A type may be an instance of multiple classes.

A class may extend another class. e.g.

1. `Fractional` extends `Num`

Outine

type: Introduction

1. Introduces a new name for an existing type (similar to `typedef` in C).
    1. the type name begins with a capital letter

Example

```
type String=[Char]
type Pos=(Int, Int)
```

type (continued)

1. type definitions cannot be recursive: e.g. `type Tree = (Int, [Tree])` is not valid
2. type declarations can be parametric!

Example of parametric type definition

```
type Pair a = (a,a)
type Assoc k v = [(k,v)]

find::Eq k=> k -> Assoc k v -> [v]
find k ts = [v | (k',v) <- ts, k==k']
```

Outine

`data` : Introduction

Using `data`, we can

1. can define new types
2. can define recursive types
3. can define parametric types

Defining a Boolean type

A Boolean type

```
data Boolean = Truth | Falsehood
```

Explanation

1. `Truth` and `Falsehood` are *constructors*.
2. `Boolean` type has only two values, `Truth` and `Falsehood`
3. `|` is called "or", which denotes that values of the type `Boolean` may be either a `Truth` or a `Falsehood`.

Constructors

1. Constructors may take arguments, like methods, - but the arguments are types, not values.
2. Constructors "construct" the values.
3. Constructors, unlike methods, have no defining equations
4. They are like "placeholders" for values
5. Constructors, unlike ordinary methods, start with capital letters

Constructors with arguments

Constructors with arguments

```
data Shape = Circle Float | Rectangle Float Float

-- try :type Circle

area::Shape->Float
area (Circle x) = 3.14 * x * x
area (Rectangle x y) = x * y
```

Parametric and Recursive data declarations

Parametric data type

```
data BinaryTree a = Nil | Node a (BinaryTree a)
                                 (BinaryTree a)
  deriving Show
```

Outine

newtype

If a type has a single constructor with a single argument, then we can use `newtype`. e.g.

newtype example

```
newtype Query = Q [Char]
```

Comparison with `type` and `data`

1. unlike `type`: `String` and `Query` are different. `Query` is not a synonym for `[Char]`. So `[Char]` and `Query` are different for the type checker.
2. unlike `data` : (Advanced) after type-checking, the constructor is "erased". So at run-time, `Query` is as efficient as `[Char]`.

Outine

Outine