Type Classes in Haskell

Generated by LectureMate

February 22, 2025

1 Introduction

In this lecture, we discussed type classes in Haskell, a powerful feature that allows us to define functions that can operate on different types. Type classes group types that share common behavior, enabling polymorphism in Haskell.

2 Understanding Type Classes

Type classes are used to describe classes of types that have something in common. For example, the ‘Eq‘ type class is used for types that support equality operations. Other examples include ‘Show‘ for types that can be converted to strings and ‘Ord‘ for types that can be ordered.

2.1 Types and Values

In Haskell, values and expressions have types. Common built-in types include:

- `Int`
- `Bool`
- Lists (e.g., `[Bool]`, `[Int]`)
- Algebraic data types (e.g., `data Tree = Empty | Node Tree Tree`)

We can define type synonyms, such as:

- `type Name = String`
- `type Predicate a = a -> Bool`

3 Type Classes in Detail

Type classes allow us to define functions that can work with any type that is an instance of that class. For example, the membership function for lists requires that the type of the elements supports equality.

3.1 Example: Membership Function

The membership function checks if a value is in a list:

```
elem :: Eq a => a -> [a] -> Bool
```

This signature indicates that the function requires its type parameter a to be an instance of the ‘Eq‘ type class.

3.2 Defining Type Classes

To define a type class, we use the following syntax:

```
class ClassName a where
    function1 :: a -> ...
    function2 :: a -> ...
```

For example, the ‘Eq‘ type class can be defined as:

```
class Eq a where
    (==) :: a -> a -> Bool
    (/=) :: a -> a -> Bool
```

3.3 Instances of Type Classes

To make a type an instance of a type class, we define the required functions. For example, to make ‘Int‘ an instance of ‘Eq‘, we can write:

```
instance Eq Int where
    x == y = ...  -- implementation using built-in equality
```

4 Common Type Classes

4.1 Eq

The ‘Eq‘ type class is used for types that support equality. It includes functions like $==$ and $/=$.

4.2 Ord

The ‘Ord‘ type class is for types that can be ordered. It includes functions like $<$, $>$, and $<=$. An instance of ‘Ord‘ must also be an instance of ‘Eq‘.

4.3 Show

The ‘Show‘ type class is for types that can be converted to strings. It includes the function `show`.

5 Examples of Type Class Instances

We can define instances for various types:

- `Bool` as an instance of `Eq`, `Ord`, and `Show`.
- Pairs as instances of `Eq` and `Ord` based on their components.
- Lists as instances of `Eq` and `Ord` recursively.

6 Automatic Derivation

Haskell provides a mechanism to automatically derive instances of certain type classes for algebraic data types using the `deriving` keyword:

```
data Bool = False | True deriving (Eq, Ord, Show)
```

7 Conclusion

Type classes in Haskell provide a powerful way to define generic functions that can operate on a variety of types. Understanding how to define and use type classes is essential for effective Haskell programming.