Discussion questions for
How to make ad-hoc polymorphism less ad hoc
COMP 150FP

September 13, 2013

Background In addition to the classes in Wadler and Blott's paper, standard Haskell defines the `Show` and `Read` type classes for converting values to and from strings. We have functions `show` and `read` with these types:

```
show :: (Show a) => a -> String
read :: (Read a) => String -> a
```

And there are *lots* of instances, including the following:

```
Show Bool
(Show a) => Show (Maybe a)
(Show a, Show b) => Show (Either a b)
(Show a) => Show [a]
Show Int
Show Integer
Show Float
Show Double
(Show a, Show b) => Show (a, b)
```

```
Read Bool
(Read a) => Read (Maybe a)
(Read a, Read b) => Read (Either a b)
(Read a) => Read [a]
Read Int
Read Integer
Read Float
Read Double
(Read a, Read b) => Read (a, b)
```

Warmup question

1. Write a class declaration for `Show` and instances for the `Maybe` and `Pair` types. I'd prefer that you use the board, not a computer, but if you want to compile your code you'll need

   ```
   import Prelude hiding (Show, show)
   ```

Type system review

2. In each of your instance declarations from question 1, label each appearance of the word `show` with its type.

3. Explain in informal English the properties of a type variable or type constructor of kind $*$. Likewise for kind $* \Rightarrow *$.

4. Give an example of a Haskell or ML type constructor of kind $*$. Likewise for kind $* \Rightarrow *$.

Paper questions

2. Both C++ and Ada enable a programmer to overload functions, procedures, methods, and operators. So what's the big deal about type classes? Are there things type classes can do that you can't do in C++ or Ada? If so, list them and give an example of each.

3. How do you imagine the Haskell community reacted to type classes? How might the implementors have reacted?

4. What limitations do you see for the method?

Technical questions

5. Type classes can be implemented by passing dictionaries. Which of the rules in Appendix A arrange for dictionaries to be passed?

6. Explain what you expect to happen if you should type each of these terms into a Haskell interpreter or compiler:

   (a) `show True`

   (b) `show 9`

   (c) `read "True"`

   (d) `show $ read "9"`

   (N.B. `f $ x` ≡ `f x` and `$` associates to the right.)

7. When you compare Mark Jones's new result (functional dependencies) with the type-class proposal of Wadler and Blott, how has the *baseline* system changed? That is, even before we add functional dependencies, what new power is in the type-class system?

Type classes in practice

There's another handout containing problems that are suitable for you to work on in pairs. Before tackling the second of these problems, work with your small group to answer this question:

8. In Scheme, it is built into the syntax of the language that a function takes zero or more arguments. In Haskell, all functions are Curried, so the number of arguments to a function is determined by the number of arrows in the type.

   Write a syntactic proof system for the judgment "τ returns a Boolean" such that for any *function* type τ, the judgment is provable if and only if τ describes a function that returns a Boolean.

   You are then ready to move on to the Programming Practice sheet.