Taming the deriving zoo

Ryan Scott

Indiana University

github.com/RyanGlScott
rgscott@indiana.edu

April 12, 2017
What deriving can do for you

Standard class instances
What deriving can do for you

Instances for newtypes
Instances for any class
Okay, what’s the problem?
Quiz time

What does this do?
Quiz time

- We can't derive Show via GeneralizedNewtypeDeriving :(  

Quiz time

What does this do?
Quiz time
The deriving resolution algorithm

1. If deriving a class which supports bespoke instances:
   1. If deriving Eq, Ord, Ix, or Bounded for a newtype, use the GeneralizedNewtypeDeriving strategy (even if the language extension isn't enabled).
   2. If deriving Functor, Foldable, or Enum for a newtype, the datatype can be successfully used with GeneralizedNewtypeDeriving, and -XGeneralizedNewtypeDeriving has been enabled, use the GeneralizedNewtypeDeriving strategy.
   3. Otherwise, if deriving a class which supports bespoke instances, and the corresponding language extension is enabled (if necessary), use the bespoke strategy. If the language extension is not enabled, throw an error.

2. If not deriving a class which supports bespoke instances:
   1. If deriving an instance for a newtype and both -XGeneralizedNewtypeDeriving and -XDeriveAnyClass are enabled, default to DeriveAnyClass, but emit a warning stating the ambiguity.
   2. Otherwise, if -XDeriveAnyClass is enabled, use DeriveAnyClass.
   3. Otherwise, if deriving an instance for a newtype, the datatype and typeclass can be successfully used with GeneralizedNewtypeDeriving, and -XGeneralizedNewtypeDeriving is enabled, do so.
   4. Otherwise, throw an error.

wat.
Solution: deriving strategies!

- Allow programmers to disambiguate the strategy they want to use when deriving
- Example:
The three (current) deriving strategies

**bespoke**

- Definition: *be-spoke* (adj.) Tailor-made, custom-built
- Derives a "hand-crafted instance" for a class
- Only applies to a handful of classes GHC knows about (Eq, Show, Functor, Data, etc.)
The three (current) deriving strategies

newtype
The three (current) deriving strategies

\texttt{anyclass}
Takeaways

- Deriving strategies resolve many current limitations and ambiguities with the deriving mechanism (including GHC Trac #10598)
- Make it easier to extend deriving in the future
- Will (hopefully) land in GHC 8.2

Any questions?
The bikeshed needs a new coat of paint

- bespoke might sound weird if you’re not used to Commonwealth English
- Other suggestions:
  - standard
  - builtin
  - magic
  - wiredin
  - native
  - original
  - specialized
Instead of this syntax:

We could also use this syntax:
We could avoid allocating new keywords in one of two ways

- Pragmas:
- Magic type synonyms:
Other possible deriving strategies

WARNING
Half-baked ideas ahead!
Unsafe GeneralizedNewtypeDeriving

- You currently can’t do this (because roles aren’t higher-order)
- We could have a variant of the newtype strategy that uses unsafeCoerce instead of coerce
Context-less StandaloneDeriving

- You currently must provide an instance context whenever you derive an instance standalone (whereas deriving clauses don't)
- Might we allow users to write standalone instances without a context?
- Could facilitate Template Haskell libraries which tackle boilerplate
GHC plugin-based deriving strategies

- Allow users to write their own strategies