Deriving-via

BALDUR BLÖNDAL,
ANDRES LÖH, Well-Typed LLP
RYAN SCOTT, Indiana University

We present a new Haskell language extension that miraculously solves all problems in generic programming that ever existed.

ACM Reference Format:
Baldur Blöndal, Andres Löh, and Ryan Scott. 2017. Deriving-via. 1, 1 (November 2017), 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

"These types we write down they're not just names for data representations in memory, they're tags that queue in mathematical structures that we exploit."[1]

1 INTRODUCTION

It is common folklore that `Monoid`s can be lifted over `Applicative`s,

```
instance (Appliative f, Monoid a) ⇒ Monoid (f a) where
  mempty :: f a
  mempty = pure mempty

  mappend :: f a → f a → f a
  mappend = liftA2 mappend
```

Conor McBride calls this "routine programming" using `Monoid` and `Applicative` as building blocks.[2]

But this instance is undesirable for multiple reasons (TODO: more reasons, rewrite)

- It overlaps with every `Monoid` instance over an applied type.
- "Structure of the `f` is often considered more significant that that of `x`."[3]
- It may not be the desired `Monoid`: Some constructors have an 'inherent monoidal structure', most notably the *free monoid* (lists: `[a]`) where we prioritize the list structure and not that of the elements.

Lists are in fact an instance of a wholly separate way of defining `Monoid`s based on `Alternative`

```
instance Alternative f ⇒ Monoid (f a) where
  mempty :: f a
  mempty = empty
```

[1]Taken from unknown position: https://www.youtube.com/watch?v=3U3lV5VPmOU
[2]http://strictlypositive.org/Idiom.pdf
[3]Much of this is stolen from Conor: https://personal.cis.strath.ac.uk/conor.mcbride/so-pigworker.pdf

Authors' addresses: Baldur Blöndal; Andres Löh, Well-Typed LLP; Ryan Scott, Indiana University.

Unpublished working draft. Not for distribution.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2017 Copyright held by the owner/author(s).
XXXX-XXXX/2017/11-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

```
  mappend :: f a -> f a -> f a
  mappend = (<|>)
```

So what are our options.

An unfortunate solution is to duplicate code

```
instance Monoid a => Monoid (IO a) where
  mempty  = pure mempty
  mappend = liftA2 mappend

instance (Monoid a, Monoid b) => Monoid (a, b) where
  mempty = pure mempty
  mappend = liftA2 mappend

instance Monoid b => Monoid (a -> b) where
  mempty = pure mempty
  mappend = liftA2 mappend
```

but this quickly becomes unviable as `Num`, `Floating` and `Fractional` which amount to around 50 methods lifted in the exact same way. Another solution provided by Conal Elliott is to use preprocessor.[4]

But we already have

2 EXAMPLES

3 FORMALISM

4 ADVANCED USES

- **Avoiding orphan instances** Before we had a `Monoid (IO a)` instance, we could not write[5]

  ```
  newtype Plugin = Plugin (IO (String -> IO ()))
    deriving Monoid
  ```

  deriving via enables us to override and insert arbitrary instances adding the following line

  ```
  via App IO (String -> App IO ())
  ```

- **Asymptotic improvement** For representable functors the definitions of `m *> _ = m` and `_ <* m = m` become *O(1)* [6]

4.1 Generalized GeneralizedNewtypeDeriving

4.2 DeriveAnyClass

5 LIMITATIONS, CONCLUSIONS AND FUTURE WORK

6 RELATED WORK

[4]https://hackage.haskell.org/package/applicative-numbers
[5]http://www.haskellforall.com/2014/07/equational-reasoning-at-scale.html
[6]Edward Kmett: https://ghc.haskell.org/trac/ghc/ticket/10892?cversion=0&cnum_hist=4#comment:4

Unpublished working draft. Not for distribution.