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 structural', 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.

XXXX-XXXX/2017/11-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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