Conor McBride: *The Derivative of a Regular Type is its Type of One-Hole Contexts*

Gergő Érdi
http://gergo.erdi.hu/

Papers We Love.SG, June 2015.

Introduction: Zippers

- Gérard Huet: *The Zipper* (Functional Pearl, 1997): functional equivalent of a pointer into a data structure: turn a tree-like structure into a subtree in context.

$$\begin{aligned} \mathbf{data}\ \mathit{Paren}\ \alpha &= \mathit{Leaf}\ \alpha \\ &\mid \mathit{Branch}\ (\mathit{Paren}\ \alpha)\ (\mathit{Paren}\ \alpha) \end{aligned}$$

$$\mathbf{type}\ \mathit{ParenZ}\ \alpha = ([\mathit{Either}\ (\mathit{Paren}\ \alpha)\ (\mathit{Paren}\ \alpha)], \mathit{Paren}\ \alpha)$$

Introduction: Zippers

- Gérard Huet: *The Zipper* (Functional Pearl, 1997): functional equivalent of a pointer into a data structure: turn a tree-like structure into a subtree in context.

$$
\begin{aligned}
&\textbf{data}\ \mathit{Paren}\ \alpha = \mathit{Leaf}\ \ \alpha \\
&\qquad\qquad\qquad\ \mid \mathit{Branch}\ (\mathit{Paren}\ \alpha)\ (\mathit{Paren}\ \alpha) \\
&\textbf{type}\ \mathit{ParenZ}\ \alpha = ([\mathit{Either}\ (\mathit{Paren}\ \alpha)\ (\mathit{Paren}\ \alpha)], \mathit{Paren}\ \alpha)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{zip} :: \mathit{ParenZ}\ \alpha \to \mathit{Paren}\ \alpha \\
&\mathit{zip}\ (\mathit{path}, t) = \mathit{foldl}\ \mathit{plug}\ t\ \mathit{path} \\
&\quad \textbf{where} \\
&\qquad \mathit{plug}\ \mathit{t2}\ (\mathit{Left}\ \mathit{t1})\ \ \ = \mathit{Branch}\ \mathit{t1}\ \ \mathit{t2} \\
&\qquad \mathit{plug}\ \mathit{t1}\ (\mathit{Right}\ \mathit{t2}) = \mathit{Branch}\ \mathit{t1}\ \ \mathit{t2}
\end{aligned}
$$

Introduction: Zippers

- Gérard Huet: *The Zipper* (Functional Pearl, 1997): functional equivalent of a pointer into a data structure: turn a tree-like structure into a subtree in context.

$$
\begin{aligned}
&\textbf{data}\ \mathit{Paren}\ \alpha = \mathit{Leaf}\ \alpha \\
&\qquad\qquad\qquad\quad \mid \mathit{Branch}\ (\mathit{Paren}\ \alpha)\ (\mathit{Paren}\ \alpha) \\
&\textbf{type}\ \mathit{ParenZ}\ \alpha = ([\mathit{Either}\ (\mathit{Paren}\ \alpha)\ (\mathit{Paren}\ \alpha)], \mathit{Paren}\ \alpha)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{holes} :: \mathit{Paren}\ \alpha \to [\mathit{ParenZ}\ \alpha] \\
&\mathit{holes}\ \mathit{Leaf}\ \{\,\} \qquad\quad = [\,] \\
&\mathit{holes}\ (\mathit{Branch}\ \mathit{t1}\ \mathit{t2}) = [([\mathit{Left}\ \mathit{t1}], \mathit{t2}), ([\mathit{Right}\ \mathit{t2}], \mathit{t1})]
\end{aligned}
$$

Is there a principled way of coming up with all this?

Is there a principled way of coming up with all this?

```
{-# LANGUAGE TypeFamilies, TypeOperators #-}
{-# LANGUAGE EmptyDataDecls, EmptyCase #-}
{-# LANGUAGE PatternSynonyms #-}
module ZipperDeriv where
import Prelude hiding (zip, unzip)
import Control.Arrow (first)
```

Sums of products

$$
\begin{array}{l}
\textbf{newtype}\ \mathit{Const}\ \alpha \quad = \mathit{Const}\ \{\, \mathit{unConst} :: \alpha \,\} \\
\textbf{type}\ \mathbf{1} = \mathit{Const}\ () \\
\textbf{pattern}\ \mathbf{1} = \mathit{Const}\ () \\
\textbf{type}\ \mathbf{0} = \mathit{Const}\ \mathit{Void} \\
\textbf{infixl}\ 6\ \oplus \\
\textbf{data}\ (\oplus)\ f\ g \quad = \mathit{InL}\ \ f \\
\qquad\qquad\qquad\quad \mid\ \mathit{InR}\ \ g \\
\textbf{infixl}\ 7\ \otimes \\
\textbf{data}\ (\otimes)\ f\ g \quad = f\ \ \otimes g
\end{array}
$$

Sums of products

$$
\begin{aligned}
&\textbf{newtype}\ \mathit{Const}\ \alpha \quad = \mathit{Const}\ \{\mathit{unConst} :: \alpha\} \\
&\textbf{type}\ \mathbf{1} = \mathit{Const}\ () \\
&\textbf{pattern}\ \mathbf{1} = \mathit{Const}\ () \\
&\textbf{type}\ \mathbf{0} = \mathit{Const}\ \mathit{Void} \\
&\textbf{infixl}\ 6 \oplus \\
&\textbf{data}\ (\oplus)\ f\ g \quad = \mathit{InL}\ \ f \\
&\qquad\qquad\qquad\quad\ \mid \mathit{InR}\ \ g \\
&\textbf{infixl}\ 7 \otimes \\
&\textbf{data}\ (\otimes)\ f\ g \quad = f \ \ \otimes g
\end{aligned}
$$

Hey look, it's a semiring! (modulo handwavy isomorphisms)

Sums of products

$$
\begin{array}{l}
\textbf{newtype}\ \mathit{Const}\ \alpha\ x = \mathit{Const}\ \{\mathit{unConst} :: \alpha\} \\
\textbf{type}\ \mathbf{1} = \mathit{Const}\ () \\
\textbf{pattern}\ \mathbf{1} = \mathit{Const}\ () \\
\textbf{type}\ \mathbf{0} = \mathit{Const}\ \mathit{Void} \\
\textbf{infixl}\ 6 \oplus \\
\textbf{data}\ (\oplus)\ f\ g\ x = \mathit{InL}\ (f\ x) \\
\qquad\qquad\qquad\quad \mid \mathit{InR}\ (g\ x) \\
\textbf{infixl}\ 7 \otimes \\
\textbf{data}\ (\otimes)\ f\ g\ x = f\ x \otimes g\ x \\
\\
\textbf{newtype}\ X\ x = X\ \{\mathit{unX} :: x\}
\end{array}
$$

Hey look, it's a semiring! (modulo handwavy isomorphisms)
So let's build polynomials!

Regular types: fixed points of polynomials

The original paper is formulated for polynomials, and fixed points, in many variables; but for simplicity's sake, this presentation uses single-variable polynomials.

$$\textbf{newtype}\ \mu\ f = \mathit{Fix}\ \{\mathit{unFix} :: f\ (\mu\ f)\}$$

Regular types: fixed points of polynomials

The original paper is formulated for polynomials, and fixed points, in many variables; but for simplicity's sake, this presentation uses single-variable polynomials.

$$\textbf{newtype}\ \mu\ f = \mathit{Fix}\ \{\mathit{unFix} :: f\ (\mu\ f)\}$$

This corresponds to *regular* data types because you can only do wholesale induction in the datatype definition: the *syntax* already only allows for defining non-parametric data types.

Examples!

$$\textbf{type}\ \mathit{Puzzle1F}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X)$$
$$\textbf{type}\ \mathit{Puzzle1}\quad \alpha = \mu\ (\mathit{Puzzle1F}\ \alpha)$$

Examples!

$$\textbf{type}\ \mathit{ListF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X)$$
$$\textbf{type}\ \mathit{List}\ \ \alpha = \mu\,(\mathit{ListF}\ \alpha)$$

$$\textbf{pattern}\ \mathit{Nil} =$$
$$\textbf{pattern}\ \mathit{Cons}\ x\ xs =$$

Examples!

$$
\begin{aligned}
&\textbf{type}\ \mathit{ListF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X)\\
&\textbf{type}\ \mathit{List}\quad \alpha = \mu\ (\mathit{ListF}\ \alpha)
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{pattern}\ \mathit{Nil} &&= \mathit{Fix}\ (\mathit{InL}\ \mathbf{1})\\
&\textbf{pattern}\ \mathit{Cons}\ x\ \mathit{xs} &&= \mathit{Fix}\ (\mathit{InR}\ (\mathit{Const}\ x \otimes X\ \mathit{xs}))
\end{aligned}
$$

Examples!

$$
\begin{aligned}
&\textbf{type}\ \mathit{ListF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X) \\
&\textbf{type}\ \mathit{List}\quad \alpha = \mu\ (\mathit{ListF}\ \alpha)
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{type}\ \mathit{Puzzle2F}\ \alpha = \mathit{Const}\ \alpha \oplus (X \otimes X) \\
&\textbf{type}\ \mathit{Puzzle2}\quad \alpha = \mu\ (\mathit{Puzzle2F}\ \alpha)
\end{aligned}
$$

Examples!

$$\textbf{type}\ \mathit{ListF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X)$$
$$\textbf{type}\ \mathit{List}\quad \alpha = \mu\,(\mathit{ListF}\ \alpha)$$

$$\textbf{type}\ \mathit{ParenF}\ \alpha = \mathit{Const}\ \alpha \oplus (X \otimes X)$$
$$\textbf{type}\ \mathit{Paren}\quad \alpha = \mu\,(\mathit{ParenF}\ \alpha)$$

$$\textbf{pattern}\ \mathit{Leaf}\ x \qquad =$$
$$\textbf{pattern}\ \mathit{Pair}\ t1\ t2 =$$

Examples!

$$\begin{aligned}
&\textbf{type } \mathit{ListF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X) \\
&\textbf{type } \mathit{List}\ \ \alpha = \mu\,(\mathit{ListF}\ \alpha)
\end{aligned}$$

$$\begin{aligned}
&\textbf{type } \mathit{ParenF}\ \alpha = \mathit{Const}\ \alpha \oplus (X \otimes X) \\
&\textbf{type } \mathit{Paren}\ \ \alpha = \mu\,(\mathit{ParenF}\ \alpha)
\end{aligned}$$

$$\begin{aligned}
&\textbf{pattern } \mathit{Leaf}\ x \quad\ \ = \mathit{Fix}\,(\mathit{InL}\,(\mathit{Const}\ x)) \\
&\textbf{pattern } \mathit{Pair}\ t1\ t2 = \mathit{Fix}\,(\mathit{InR}\,(X\ t1 \otimes X\ t2))
\end{aligned}$$

Examples!

$$\begin{aligned}
&\textbf{type } \mathit{ListF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X) \\
&\textbf{type } \mathit{List}\ \ \ \ \alpha = \mu\ (\mathit{ListF}\ \alpha)
\end{aligned}$$

$$\begin{aligned}
&\textbf{type } \mathit{ParenF}\ \alpha = \mathit{Const}\ \alpha \oplus (X \otimes X) \\
&\textbf{type } \mathit{Paren}\ \ \ \ \alpha = \mu\ (\mathit{ParenF}\ \alpha)
\end{aligned}$$

$$\begin{aligned}
&\textbf{type } \mathit{Puzzle3F}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X \otimes X) \\
&\textbf{type } \mathit{Puzzle3}\ \ \ \ \alpha = \mu\ (\mathit{Puzzle3F}\ \alpha)
\end{aligned}$$

Examples!

$$\textbf{type } \mathit{ListF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X)$$
$$\textbf{type } \mathit{List}\ \ \alpha = \mu\,(\mathit{ListF}\ \alpha)$$

$$\textbf{type } \mathit{ParenF}\ \alpha = \mathit{Const}\ \alpha \oplus (X \otimes X)$$
$$\textbf{type } \mathit{Paren}\ \ \alpha = \mu\,(\mathit{ParenF}\ \alpha)$$

$$\textbf{type } \mathit{BTreeF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X \otimes X)$$
$$\textbf{type } \mathit{BTree}\ \ \alpha = \mu\,(\mathit{BTreeF}\ \alpha)$$

$$\textbf{pattern } \mathit{Empty} =$$
$$\textbf{pattern } \mathit{Node}\ x\ t1\ \ t2 =$$

Examples!

$$\textbf{type}\ \mathit{ListF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X)$$
$$\textbf{type}\ \mathit{List}\ \ \alpha = \mu\ (\mathit{ListF}\ \alpha)$$

$$\textbf{type}\ \mathit{ParenF}\ \alpha = \mathit{Const}\ \alpha \oplus (X \otimes X)$$
$$\textbf{type}\ \mathit{Paren}\ \ \alpha = \mu\ (\mathit{ParenF}\ \alpha)$$

$$\textbf{type}\ \mathit{BTreeF}\ \alpha = \mathbf{1} \oplus (\mathit{Const}\ \alpha \otimes X \otimes X)$$
$$\textbf{type}\ \mathit{BTree}\ \ \alpha = \mu\ (\mathit{BTreeF}\ \alpha)$$

$$\textbf{pattern}\ \mathit{Empty} = \mathit{Fix}\ (\mathit{InL}\ \mathbf{1})$$
$$\textbf{pattern}\ \mathit{Node}\ x\ \mathit{t1}\ \mathit{t2} = \mathit{Fix}\ (\mathit{InR}\ (\mathit{Const}\ x \otimes X\ \mathit{t1} \otimes X\ \mathit{t2}))$$

What do we expect of a type for holes $\partial\, f$?

Without even defining what exactly we mean by a hole

What do we expect of a type for holes $\partial\ f$?

Without even defining what exactly we mean by a hole

- $\partial\ (f \oplus g)$: Either we have an *InL* value with a hole, or an *InR* value with a hole: $\partial\ f \oplus \partial\ g$
- $\partial\ (f \otimes g)$: Either we have a hole in the first component (and the second is untouched), or the hole is in the second component: $(\partial\ f \otimes g) \oplus (f \otimes \partial\ g)$

What do we expect of a type for holes $\partial\ f$?

We need to be more specific on what a hole is to continue.
So let's say we want to be able to

- Get all holes into which we can plug a subtree (of the same type as the original type)
- Plug a subtree into a hole

$$
\begin{array}{l}
\textbf{class } \mathit{Diffable}\ (f :: * \to *)\ \textbf{where} \\
\quad \textbf{type } \partial\ f :: * \to * \\
\quad \mathit{holes}' :: f\ x \to [(\partial\ f\ x, x)] \\
\quad \mathit{plug}' :: \partial\ f\ x \to x \to f\ x
\end{array}
$$

What do we expect of a type for holes $\partial\ f$?

We need to be more specific on what a hole is to continue.
So let's say we want to be able to

- Get all holes into which we can plug a subtree (of the same type as the original type)
- Plug a subtree into a hole

$$
\begin{aligned}
&\textbf{class } \mathit{Diffable}\ (f :: * \to *)\ \textbf{where} \\
&\quad \textbf{type } \partial\ f :: * \to * \\
&\quad \mathit{holes}' :: f\ x \to [(\partial\ f\ x, x)] \\
&\quad \mathit{plug}' :: \partial\ f\ x \to x \to f\ x
\end{aligned}
$$

This should give us an idea of what to do for $\mathit{Const}\ \alpha$ and X:

- $\mathit{Const}\ \alpha\ x$ has no possible positions (of type x): $\partial\ (\mathit{Const}\ \alpha) = \mathbf{0}$
- $X\ x$ has exactly one position of type x (no further labelling needed): $\partial\ X = \mathbf{1}$

Looks familiar?

$$\textbf{type}\ \partial\ (Const\ \alpha) = \mathbf{0}$$
$$\textbf{type}\ \partial\ X = \mathbf{1}$$
$$\textbf{type}\ \partial\ (f \oplus g) = \partial\ f \oplus \partial\ g$$
$$\textbf{type}\ \partial\ (f \otimes g) = (\partial\ f \otimes g) \oplus (f \otimes \partial\ g)$$

Looks familiar?

$$
\begin{aligned}
&\textbf{type}\ \partial\ (\mathit{Const}\ \alpha) = \mathbf{0} \\
&\textbf{type}\ \partial\ X = \mathbf{1} \\
&\textbf{type}\ \partial\ (f \oplus g) = \partial\ f \oplus \partial\ g \\
&\textbf{type}\ \partial\ (f \otimes g) = (\partial\ f \otimes g) \oplus (f \otimes \partial\ g)
\end{aligned}
$$

Surprise! Turns out using the names *Diffable* and ∂ (and "Derivative" in the paper's title) was a reasonable choice!

Looks familiar?

$$
\begin{aligned}
&\textbf{type}\ \partial\ (\mathit{Const}\ \alpha) = \mathbf{0} \\
&\textbf{type}\ \partial\ X = \mathbf{1} \\
&\textbf{type}\ \partial\ (f \oplus g) = \partial\ f \oplus \partial\ g \\
&\textbf{type}\ \partial\ (f \otimes g) = (\partial\ f \otimes g) \oplus (f \otimes \partial\ g)
\end{aligned}
$$

Surprise! Turns out using the names *Diffable* and ∂ (and "Derivative" in the paper's title) was a reasonable choice!

See the source code of the slides for the full implementation of the *Diffable* typeclass for $\mathit{Const}\ \alpha$, X, $f \oplus g$ and $f \otimes g$.

Making a zipper from holes

We now have a way of taking apart *one level* of an inductive data structure by having a type $\partial\, f$ which has a *hole* in it.

Making a zipper from holes

We now have a way of taking apart *one level* of an inductive data structure by having a type $\partial\ f$ which has a *hole* in it.
We can turn this into a type $Zipper\ f$ (a zipper for $\mu\ f$) by repeatedly choosing a hole and putting either one more level of data structre into it, or finishing with a $\mu\ f$:

$$\textbf{type}\ D\ f = \partial\ f\ (\mu\ f)$$

$$holes :: (Diffable\ f) \Rightarrow \mu\ f \rightarrow [(D\ f, \mu\ f)]$$
$$holes = holes' \circ unFix$$

$$plug :: (Diffable\ f) \Rightarrow D\ f \rightarrow \mu\ f \rightarrow \mu\ f$$
$$plug\ df\ = Fix \circ plug'\ df$$

$$\textbf{type}\ Zipper\ f = ([D\ f], \mu\ f)$$

$$zip :: (Diffable\ f) \Rightarrow Zipper\ f \rightarrow \mu\ f$$
$$zip\ (path, t) = foldl\ (flip\ plug)\ t\ path$$

$$unzip :: (Diffable\ f) \Rightarrow Zipper\ f \rightarrow [Zipper\ f]$$
$$unzip\ (dfs, t) = map\ (first\ (:dfs))\ (holes\ t)$$

Example: all zippers of a parenthesization

Let's enumerate all possible zippers for a given container (by recursively calling *unzip*):

$$
\begin{aligned}
&\mathit{unzips} :: (\mathit{Diffable}\ f) \Rightarrow \mu\ f \rightarrow [\mathit{Zipper}\ f] \\
&\mathit{unzips}\ t = \mathit{go}\ ([\,], t) \\
&\quad \textbf{where} \\
&\qquad \mathit{go}\ z = z : \mathit{concatMap}\ \mathit{go}\ (\mathit{unzip}\ z)
\end{aligned}
$$

We are going to use this to enumerate all zippers of this type:

$$
\begin{aligned}
&\textbf{type}\ \mathit{Paren}\ \alpha && = \mu\ (\mathit{Const}\ \alpha \oplus (X \otimes X)) \\
&\textbf{pattern}\ \mathit{Leaf}\ x && = \mathit{Fix}\ (\mathit{InL}\ (\mathit{Const}\ x)) \\
&\textbf{pattern}\ \mathit{Pair}\ \mathit{t1}\ \mathit{t2} && = \mathit{Fix}\ (\mathit{InR}\ (X\ \mathit{t1} \otimes X\ \mathit{t2}))
\end{aligned}
$$

Example: all zippers of a parenthesization

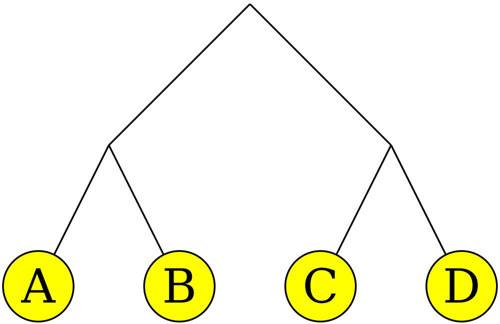
Replace with a marker every possible pointed subtree:

$$
\begin{array}{l}
\mathit{probe} :: \alpha \to \mathit{Paren}\ \alpha \to [\mathit{Paren}\ \alpha] \\
\mathit{probe}\ \mathit{marker} = \mathit{map}\ \mathit{mark} \circ \mathit{unzips} \\
\quad \textbf{where} \\
\qquad \mathit{mark}\ (z, t) = \mathit{zip}\ (z, \mathit{Leaf}\ \mathit{marker})
\end{array}
$$

Example: all zippers of a parenthesization

An example parenthesization:

$$p :: \mathit{Paren}\ (\mathit{Maybe}\ \mathit{Char})$$
$$p = (\mathit{Leaf}\ (\mathit{Just}\ \texttt{'A'})\ `\mathit{Pair}`\ \mathit{Leaf}\ (\mathit{Just}\ \texttt{'B'}))\ `\mathit{Pair}`\ (\mathit{Leaf}\ (\mathit{Just}\ \texttt{'C'})$$



Example: all zippers of a parenthesization

An example parenthesization:

$$p :: \mathit{Paren}\ (\mathit{Maybe}\ \mathit{Char})$$
$$p = (\mathit{Leaf}\ (\mathit{Just}\ \texttt{'A'})\ `\mathit{Pair}`\ \mathit{Leaf}\ (\mathit{Just}\ \texttt{'B'}))\ `\mathit{Pair}`\ (\mathit{Leaf}\ (\mathit{Just}\ \texttt{'C'})$$