General elimination and catamorphisms

David Ripley

Monash University
http://davewripley.rocks

The question

Gentzen 1934:

"The introductions represent, as it were, the 'definitions' of the symbols concerned, and the eliminations are no more… than the consequences of these definitions….

By making these ideas more precise it should be possible to display the E-inferences as unique functions of their corresponding I-inferences"

"Such [introduction] laws will be 'self-justifying': we are entitled simply to stipulate [them], because by so doing we fix...the meanings of the logical constants that they govern"

"Plainly, the elimination rules are not consequences of the introduction rules in the straightfoward sense of being derivable from them; Gentzen must therefore have had in mind some more powerful means of drawing consequences"

"Plainly, the elimination rules are not consequences of the introduction rules in the straightfoward sense of being derivable from them; Gentzen must therefore have had in mind some more powerful means of drawing consequences"

What is this “more powerful means”?

Given introduction rules for a connective,
how to determine its elimination rules?

Examples

A proposition is either:

- atomic p, q, r, . . ., or
- $A \wedge B$, where A and B are propositions, or
- $A \vee B$, where A and B are propositions, or ...

A type is either:

- basic `Int`, `Char`, `Double`, ..., or
- `Pair a b`, where `a` and `b` are types, or
- `Either a b`, where `a` and `b` are types, or ...

$$\vee\mathrm{I}_l: \quad \frac{A}{A \vee B} \qquad \vee\mathrm{I}_r: \quad \frac{B}{A \vee B}$$

```haskell
data Either a b =
  Left a | Right b
```

$$\vee\mathrm{E}: \quad \frac{A \vee B \qquad \begin{matrix}(A)\\ \vdots \\ C\end{matrix} \qquad \begin{matrix}(B)\\ \vdots \\ C\end{matrix}}{C}$$

```haskell
either ::
  (a -> c) -> (b -> c)
    -> Either a b -> c
```

$$\vee\mathrm{I}_l: \quad \frac{A}{A \vee B} \qquad \vee\mathrm{I}_r: \quad \frac{B}{A \vee B}$$

```
data Either a b =
  Left a | Right b
```

$$\vee\mathrm{E}: \quad \frac{A \vee B \qquad \begin{array}{c}(A)\\ \vdots \\ C\end{array} \qquad \begin{array}{c}(B)\\ \vdots \\ C\end{array}}{C}$$

```
either ::
  (a -> c) -> (b -> c)
    -> Either a b -> c
```

$\vee\mathrm{I}_l$: $\dfrac{A}{A \vee B}$ $\vee\mathrm{I}_r$: $\dfrac{B}{A \vee B}$

$$\vee\mathrm{E}: \quad \frac{A \vee B \qquad \begin{matrix}(A)\\ \vdots \\ C\end{matrix} \qquad \begin{matrix}(B)\\ \vdots \\ C\end{matrix}}{C}$$

```haskell
data Either a b =
  Left a | Right b
```

```haskell
either ::
  (a -> c) -> (b -> c)
    -> Either a b -> c
```

$$\vee\mathrm{E}: \dfrac{\vee\mathrm{I}_l: \dfrac{\begin{matrix}\mathtt{x}\\ A\end{matrix}}{A \vee B} \qquad \begin{matrix}(A)\\ \mathtt{f}\\ C\end{matrix} \qquad \begin{matrix}(B)\\ \vdots \\ C\end{matrix}}{C}$$

$$\rightsquigarrow$$

$$\begin{matrix}\mathtt{x}\\ A\\ \mathtt{f}\\ C\end{matrix}$$

```
either ::
  (a -> c) -> (b -> c)
    -> Either a b -> c

either f _ (Left x)  = f x
```

$$\vee E: \dfrac{\vee I_l: \dfrac{\begin{matrix}\mathtt{x}\\ A\end{matrix}}{A \vee B} \qquad \begin{matrix}(A)\\ \mathtt{f}\\ C\end{matrix} \qquad \begin{matrix}(B)\\ \vdots\\ C\end{matrix}}{C}$$

$$\rightsquigarrow$$

$$\begin{matrix}\mathtt{x}\\ A\\ \mathtt{f}\\ C\end{matrix}$$

```
either ::
  (a -> c) -> (b -> c)
    -> Either a b -> c

either f _ (Left x)  = f x
```

$$
\begin{array}{ll}
\vee I_r: & \dfrac{\genfrac{}{}{0pt}{}{\texttt{y}}{B}}{A \vee B} \\
\vee E: &
\end{array}
\dfrac{\dfrac{\texttt{y}}{B}\qquad \begin{matrix}(A)\\ \vdots \\ C\end{matrix} \qquad \begin{matrix}(B)\\ \texttt{g} \\ C\end{matrix}}{C}
$$

$$\rightsquigarrow$$

$$\begin{matrix}\texttt{y}\\ B \\ \texttt{g} \\ C\end{matrix}$$

```
either ::
  (a -> c) -> (b -> c)
    -> Either a b -> c

either f _ (Left x)  = f x

either _ g (Right y) = g y
```

compound formula	compound type
I rule	type constructor
E rule	type signature for elimination function
reduction step	definition of elimination function

$$\wedge\text{I}: \quad \frac{A \qquad B}{A \wedge B}$$

```
data Pair a b = Pr a b
```

$$\wedge\text{GE}: \quad \frac{A \wedge B \qquad \begin{matrix}\underbrace{(A),(B)} \\ \vdots \\ C\end{matrix}}{C}$$

```
uncurry ::
  (a -> b -> c) -> Pair a b -> c
```

$$\wedge GE: \dfrac{\wedge I: \dfrac{\overset{\mathtt{x}}{A} \qquad \overset{\mathtt{y}}{B}}{A \wedge B} \qquad \begin{matrix}\underbrace{(A),(B)}_{\mathtt{f}} \\ C\end{matrix}}{C}$$

$$\rightsquigarrow$$

$$\begin{matrix}\underbrace{\overset{\mathtt{x}}{A} \qquad \overset{\mathtt{y}}{B}}_{\mathtt{f}} \\ C\end{matrix}$$

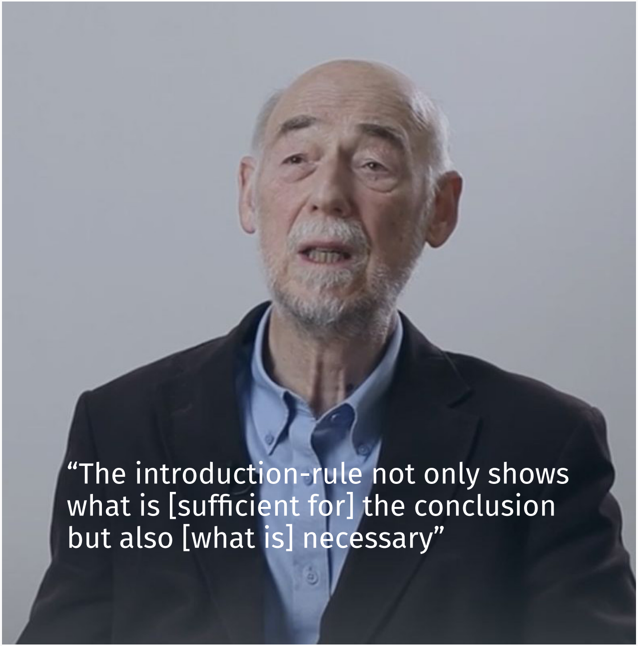
```
uncurry ::
  (a -> b -> c) -> Pair a b -> c

uncurry f (Pr x y) = f x y
```

Method 1: general elimination

“[W]hatever follows from the sufficient grounds for deriving a formula must follow from that formula...

[W]hatever follows from a formula must follow from the sufficient grounds for deriving the formula”


“The introduction-rule not only shows
what is [sufficient for] the conclusion
but also [what is] necessary”

Moriconi & Tesconi:

“[I]t is quite natural to ask what consequences can be drawn from A, given that A can be produced *only* by [certain] rules. The answer is: we can draw all the consequences that we can draw from the premisses of those rules”

$$\wedge\mathrm{I}: \quad \frac{A \qquad B}{A \wedge B}$$

$$\wedge\mathrm{GE}: \quad \frac{A \wedge B \qquad \begin{array}{c} \underbrace{(A), (B)} \\ \vdots \\ C \end{array}}{C}$$

$$\vee\mathrm{I}_l: \quad \frac{A}{A \vee B} \qquad \vee\mathrm{I}_r: \quad \frac{B}{A \vee B}$$

$$\vee\mathrm{E}: \quad \frac{A \vee B \qquad \begin{array}{c} (A) \\ \vdots \\ C \end{array} \qquad \begin{array}{c} (B) \\ \vdots \\ C \end{array}}{C}$$

Method 2: Catamorphisms

Catamorphisms give a different way to generate `either`, `uncurry`, and the like

The first step into so-called ‘recursion schemes’

Nil: $\dfrac{}{[A]}$ Cons: $\dfrac{A \qquad [A]}{[A]}$

```
data [a] = [] | a : [a]
```

In Dummett's terminology, Cons is:
pure, simple, direct,
not sheer, not single-ended,
violates the complexity condition

Nil: $\dfrac{}{[A]}$ Cons: $\dfrac{A \qquad [A]}{[A]}$

```
data [a] = [] | a : [a]
```

In Dummett's terminology, Cons is:
pure, simple, direct,
not sheer, not single-ended,
violates the complexity condition

Nil: $\dfrac{}{[A]}$ Cons: $\dfrac{A \qquad [A]}{[A]}$

```
data [a] = [] | a : [a]
```

Nil: $\dfrac{}{[A]}$ Cons: $\dfrac{A \qquad [A]}{[A]}$

```
data [a] = [] | a : [a]
```

foldr: $\dfrac{[A] \qquad B \qquad \begin{array}{c}\overbrace{(A),(B)}\\ \vdots \\ B\end{array}}{B}$

```
foldr :: (a -> b -> b) ->
  b -> [a] -> b
```

Nil: $\dfrac{}{[A]}$ Cons: $\dfrac{A \qquad [A]}{[A]}$

```
data [a] = [] | a : [a]
```

foldr: $\dfrac{[A] \qquad B \qquad \begin{array}{c}\overbrace{(A),(B)}\\ \vdots \\ B\end{array}}{B}$

```
foldr :: (a -> b -> b) ->
  b -> [a] -> b
```

Nil: $\dfrac{}{[A]}$ Cons: $\dfrac{A \qquad [A]}{[A]}$

```
data [a] = [] | a : [a]
```

foldr: $\dfrac{[A] \qquad B \qquad \begin{array}{c}\underbrace{(A), (B)} \\ \vdots \\ B\end{array}}{B}$

```
foldr :: (a -> b -> b) ->
  b -> [a] -> b
```

$$\text{foldr: } \dfrac{\text{Nil: } \dfrac{}{[A]} \qquad \begin{array}{c}\texttt{z}\\ B\end{array} \qquad \begin{array}{c}\underbrace{(A),(B)}\\ \vdots \\ B\end{array}}{B}$$

$$\Big\downarrow\!\!\wr$$

$$\begin{array}{c}\texttt{z}\\ B\end{array}$$

```
foldr :: (a -> b -> b) ->
  b -> [a] -> b

foldr _ z [] = z
```

$$\text{foldr: }\dfrac{\text{Cons: }\dfrac{\overset{\texttt{x}}{A}\qquad\overset{\texttt{xs}}{[A]}}{[A]}\qquad\overset{\texttt{z}}{B}\qquad\overset{\overbrace{(A),(B)}}{\overset{\texttt{f}}{B}}}{B}$$

$$\rightsquigarrow$$

$$\underbrace{\overset{\texttt{x}}{A}\qquad\text{foldr: }\dfrac{\overset{\texttt{xs}}{[A]}\qquad\overset{\texttt{z}}{B}\qquad\overset{\overbrace{(A),(B)}}{\overset{\texttt{f}}{B}}}{B}}_{\overset{\texttt{f}}{B}}$$

```
foldr :: (a -> b -> b) ->
  b -> [a] -> b

foldr _ z [] = z

foldr f z (x : xs) =
  f x (foldr f z xs)
```

Meijer, Fokkinga, Paterson:

“Countless list processing functions are readily recognizable as catamorphisms”

```
            ns = 1 : (3 : (5 : (7 : [])))

foldr (+) 0 ns = 1 + (3 + (5 + (7 + 0 ))) = sum ns

foldr (*) 1 ns = 1 * (3 * (5 * (7 * 1 ))) = product ns
```

```
append :: [a] -> [a] -> [a]
append xs ys = foldr (:) ys xs

concat :: [[a]] -> [a]
concat xss   = foldr append [] xss
```

$$\text{foldr:} \quad \dfrac{[A] \qquad \text{Nil:} \; \dfrac{}{[B]} \qquad \text{Cons:} \; \dfrac{\begin{matrix}(A)\\ \vdots \\ B\end{matrix} \qquad ([B])}{[B]}}{[B]}$$

$$\text{foldr: } \dfrac{\text{Nil: } \dfrac{}{[A]} \qquad \text{Nil: } \dfrac{}{[B]} \qquad \text{Cons: } \dfrac{\begin{array}{c}(A)\\ \vdots \\ B\end{array} \qquad ([B])}{[B]}}{[B]}$$

$$\Big\downarrow$$

$$\text{Nil: } \dfrac{}{[B]}$$

$$\text{foldr: } \dfrac{\text{Cons: } \dfrac{\begin{matrix}\texttt{x}\\ A\end{matrix} \qquad \begin{matrix}\texttt{xs}\\ [A]\end{matrix}}{[A]} \qquad \text{Nil: } \dfrac{}{[B]} \qquad \text{Cons: } \dfrac{\begin{matrix}(A)\\ \texttt{f}\\ B\end{matrix} \qquad ([B])}{[B]}}{[B]}$$

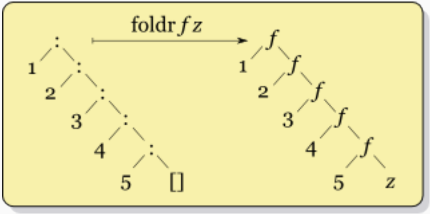
$$\rightsquigarrow$$

$$\text{Cons: } \dfrac{\begin{matrix}\texttt{x}\\ A\\ \texttt{f}\\ B\end{matrix} \qquad \text{foldr: } \dfrac{\begin{matrix}\texttt{xs}\\ [A]\end{matrix} \qquad \text{Nil: } \dfrac{}{[B]} \qquad \text{Cons: } \dfrac{\begin{matrix}(A)\\ \texttt{f}\\ B\end{matrix} \qquad ([B])}{[B]}}{[B]}}{[B]}$$

Nil: $\dfrac{}{[A]}$ Cons: $\dfrac{A \qquad [A]}{[A]}$

This is a language for building proofs of $[A]$

We can translate this language into a language for proofs of B if we can translate each rule



$$\vee I_l: \quad \frac{A}{A \vee B} \qquad \vee I_r: \quad \frac{B}{A \vee B}$$

To get a catamorphism from $A \vee B$ to C,
provide proofs `f` from A to C and `g` from B to C

Replace $\vee I_l$ with `f` and $\vee I_r$ with `g`

```
either :: (a -> c) -> (b -> c) -> Either a b -> c
either f _ (Left x)  = f x
either _ g (Right y) = g y
```

$$\wedge\text{I}: \quad \frac{A \qquad B}{A \wedge B}$$

To get a catamorphism from $A \wedge B$ to C, provide a proof `f` from A and B to C

Replace $\wedge$I with `f`

```
uncurry :: (a -> b -> c) -> Pair a b -> c
uncurry f (Pr x y) = f x y
```

Differences

General elimination and catamorphisms
have a lot in common!

Catamorphisms for disjunction
give us the usual (general) elimination rule

And catamorphisms for conjunction
give us the general elimination rule

$$[\,]\mathrm{GE}: \quad \dfrac{[A] \qquad B \qquad \begin{array}{c}\overbrace{(A),([A])} \\ \vdots \\ B\end{array}}{B}$$

$$\mathrm{foldr}: \quad \dfrac{[A] \qquad B \qquad \begin{array}{c}\overbrace{(A),(B)} \\ \vdots \\ B\end{array}}{B}$$

$$\text{Nil: } \frac{}{[A]} \qquad \text{[ ]GE: } \dfrac{[A] \quad \overset{\mathtt{z}}{B} \quad \overset{\overbrace{(A),([A])}}{\overset{\vdots}{B}}}{B} \qquad\qquad \text{Nil: } \frac{}{[A]} \qquad \text{foldr: } \dfrac{[A] \quad \overset{\mathtt{z}}{B} \quad \overset{\overbrace{(A),(B)}}{\overset{\vdots}{B}}}{B}$$

$$\rightsquigarrow \qquad\qquad\qquad\qquad \rightsquigarrow$$

$$\overset{\mathtt{z}}{B} \qquad\qquad\qquad\qquad \overset{\mathtt{z}}{B}$$

$$[\,]\text{GE}: \dfrac{\text{Cons}: \dfrac{\overset{\mathtt{x}}{A} \qquad \overset{\mathtt{xs}}{[A]}}{[A]} \qquad \overset{\mathtt{z}}{B} \qquad \overbrace{\overset{\mathtt{f}}{B}}^{(A),([A])}}{B}
\qquad\qquad
\text{foldr}: \dfrac{\text{Cons}: \dfrac{\overset{\mathtt{x}}{A} \qquad \overset{\mathtt{xs}}{[A]}}{[A]} \qquad \overset{\mathtt{z}}{B} \qquad \overbrace{\overset{\mathtt{f}}{B}}^{(A),(B)}}{B}$$

$$\rightsquigarrow \qquad\qquad\qquad \rightsquigarrow$$

$$\underbrace{\overset{\mathtt{x}}{A} \qquad \overset{\mathtt{xs}}{[A]}}_{\overset{\mathtt{f}}{B}}
\qquad\qquad
\underbrace{\overset{\mathtt{x}}{A} \qquad \text{foldr}: \dfrac{\overset{\mathtt{xs}}{[A]} \qquad \overset{\mathtt{z}}{B} \qquad \overbrace{\overset{\mathtt{f}}{B}}^{(A),(B)}}{B}}_{\overset{\mathtt{f}}{B}}$$

"The fact that the [rules for a connective] are in harmony … shows only that we draw no consequences its meaning does not entitle us to draw.

It does not show that we fully exploit that meaning"

Jacinto & Read 2017:

"[T]he E-rules should not merely be justified by the meaning conferred by the I-rules; they should allow one to infer everything that is warranted by that meaning"

So which of these approaches
allows us to capture more?

```
safeTail :: [a] -> [a]
safeTail []       = []
safeTail (x : xs) = xs
```

```
safeTail [1, 2] = [2]        safeTail [1, 7] = [7]
safeTail    [2] = []         safeTail    [7] = []
```

if `safeTail = foldr f z`, then

`safeTail (x : xs) = f x (safeTail xs)`

But `f 1 []` can't be both `[2]` and `[7]`!

So `safeTail` is not `foldr f z` for any `f`, `z`

General elimination gives us `safeTail`:

$$[\,]\mathrm{GE}: \quad \frac{[A] \qquad \mathrm{Nil}: \dfrac{}{[A]} \qquad ([A])}{[A]}$$

So catamorphisms don't fully capture general elimination

If we have any proof $\texttt{f}$ from $[A]$ to B,
we can make it into a GE:

$$\text{Nil: } \frac{}{[A]} \qquad \text{Cons: } \frac{(A) \qquad ([A])}{[A]}$$

$$[\,]\text{GE: } \frac{\begin{matrix} \texttt{xs} \\ [A] \end{matrix} \qquad \begin{matrix} \texttt{f} \\ B \end{matrix} \qquad \begin{matrix} \texttt{f} \\ B \end{matrix}}{B}$$

So GE is in some sense universal:
any proof we’re able to rig up,
we can embed in an equivalent GE

This isn’t true of foldr,
as we’ve seen

And for a given list, we can rig up something with [ ]GE that does what (foldr f z) would do with the list

But can't do it once for all lists:
foldr is recursive and [ ]GE ain't

What we rig up will include a number of [ ]GEs depending on the length of the list

So we can't give a single definition of
`append` or `concat`,

or of mapping a proof (even a fixed proof) from A to B
over an arbitrary $[A]$ to get the resulting $[B]$.

Even given `0` and `+` or `1` and `*`,
we can't define `sum` or `product` with GE alone.

foldr and [ ]GE differ

We want the strongest thing we can get,
but neither is stronger than the other

We need a hybrid!

```
foldr  :: (a ->          b -> b) -> b -> [a] -> b
listGE :: (a -> [a] ->      b) -> b -> [a] -> b
para   :: (a -> [a] -> b -> b) -> b -> [a] -> b

foldr  _ z []     = z
listGE _ z []     = z
para   _ z []     = z

foldr  f z (x:xs) = f x    (foldr f z xs)
listGE g z (x:xs) = g x xs
para   h z (x:xs) = h x xs (para  h z xs)
```

$$[\,]\mathrm{GE}: \quad \frac{[A] \qquad B \qquad \begin{array}{c} \underbrace{(A), ([A])} \\ \vdots \\ B \end{array}}{B}$$

$$\mathrm{foldr}: \quad \frac{[A] \qquad B \qquad \begin{array}{c} \underbrace{(A), (B)} \\ \vdots \\ B \end{array}}{B}$$

$$\mathrm{para}: \quad \frac{[A] \qquad B \qquad \begin{array}{c} \underbrace{(A), ([A]), (B)} \\ \vdots \\ B \end{array}}{B}$$

$$\text{para: } \dfrac{\text{Cons: } \dfrac{\overset{\texttt{x}}{A} \qquad \overset{\texttt{xs}}{[A]}}{[A]} \qquad \overset{\texttt{z}}{B} \qquad \overbrace{\overset{\texttt{h}}{B}}^{(A),\,([A]),\,(B)}}{B}$$

$$\rightsquigarrow$$

$$\underbrace{\overset{\texttt{x}}{A} \qquad \overset{\texttt{xs}}{[A]} \qquad \text{para: } \dfrac{\overset{\texttt{xs}}{[A]} \qquad \overset{\texttt{z}}{B} \qquad \overbrace{\overset{\texttt{h}}{B}}^{(A),\,([A]),\,(B)}}{B}}_{\overset{\texttt{h}}{B}}$$

Upshots

Nil: $\dfrac{}{[A]}$ Cons: $\dfrac{A \qquad [A]}{[A]}$

General elimination focuses on the grounds for introduction

$$[\,]\mathrm{GE}:\quad \dfrac{[A] \qquad B \qquad \begin{array}{c}\underbrace{(A),([A])} \\ \vdots \\ B\end{array}}{B}$$

$$\text{Nil:} \quad \frac{}{[A]} \qquad \text{Cons:} \quad \frac{A \qquad [A]}{[A]}$$

Catamorphisms focus on translating the introduction rules

$$\text{foldr:} \quad \frac{[A] \qquad B \qquad \begin{array}{c} \underbrace{(A),(B)} \\ \vdots \\ B \end{array}}{B}$$

Nil: $\dfrac{}{[A]}$ Cons: $\dfrac{A \qquad [A]}{[A]}$

Paramorphisms take both into account,
using the grounds both translated and untranslated

para: $\dfrac{[A] \qquad B \qquad \begin{matrix} \underbrace{(A), ([A]), (B)} \\ \vdots \\ B \end{matrix}}{B}$

THE
THE GOSPEL
ACCORDING TO
JOHN

KING JAMES II VERSION

CHAPTER 1

[1]In *the* beginning was the Word, and the Word was with God, and the Word was God. [2]He was in *the* beginning with God. [3]All things came into being through Him, and without Him not even one *thing* came into being that has come into being. [4]In Him was life, and the life was the light of men. [5]and the light shines in the darkness, and the darkness did not overtake it.

[6]There was a man sent from God, his name *was* John. [7]He came for a witness, that he might witness concerning the Light, that all might believe through Him. [8]He was not that Light, but that he might witness concerning the Light. [9]He was the true Light; He enlightens every

ΕΥΑΓΓΕΛΙΟΝ
ΤΟ ΚΑΤΑ ΙΩΑΝΝΗΝ

CHAPTER 1

1722 746 2258 3056 3056 2258 4314 2316
1 Ἐν ἀρχῇ ἦν ὁ λόγος, καὶ ὁ λόγος ἦν πρὸς τὸν Θεόν, καὶ
In (the) beginning was the Word, and the Word was with — God, and
2316/2258 3056 3778/2258/1722/746 4314 2316 3956
2 Θεὸς ἦν ὁ λόγος. οὗτος ἦν ἐν ἀρχῇ πρὸς τὸν Θεόν. πάντα
3 **God was the Word. This One was in beginning with God. All things**
1223 846 1096 5565 846 1096 3761/1520 1096
δι' αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο οὐδὲ ἓν ὃ γέγονεν.
through Him came into and without Him came into not even one that came into
1722 846 2222 being. 2222/2258 being (thing) 444 being.
5457 ἐν αὐτῷ ζωὴ ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀνθρώπων, καὶ
4 **In Him life was, and the life was the light — of men, and**
5 5457/1722 4653 5316 4653 846/3756/2638
τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.
the light in the darkness shines, and the darkness it not did overtake
1096 444 649 3814 2316 3686 846
6 ἐγένετο ἄνθρωπος ἀπεσταλμένος παρὰ Θεοῦ, ὄνομα αὐτῷ
There was a man having been sent from God, name to him,
2491 3778 2064 1519 3141 2443 3140 4012
7 Ἰωάννης. οὗτος ἦλθεν εἰς μαρτυρίαν, ἵνα μαρτυρήσῃ περὶ
John; this one came for a witness, that he might witness about
5457 2443 3956 4100 1223 846 3756/2258 1565
8 τοῦ φωτός, ἵνα πάντες πιστεύσωσι δι' αὐτοῦ. οὐκ ἦν ἐκεῖνος
the light, that all might believe through Him. not He was that
5457 235/2443 3140 4012 5457 2258 5457
9 τὸ φῶς, ἀλλ' ἵνα μαρτυρήσῃ περὶ τοῦ φωτός. ἦν τὸ φῶς τὸ
light, but that he might witness about the light. He was the light
228 3739 5461 3956 444 2064 1519
ἀληθινόν, ὃ φωτίζει πάντα ἄνθρωπον ἐρχόμενον εἰς τὸν
true, which enlightens every man coming into the
2889 1722 2889 2258 2889 1223 846 1096

This allows us to define things out of reach
of both [ ]GE and foldr individually

Eg the proof from $[A]$ to $[[A]]$
that replaces each member of the original list
with the list of its followers

$$\text{para:}\ \frac{[A] \qquad \text{Nil:}\ \dfrac{}{[[A]]} \qquad \text{Cons:}\ \dfrac{([A]) \qquad ([[A]])}{[[A]]}}{[[A]]}$$

Question:

Can para do anything that foldr and $[\,]$GE can't do together?

Known (Meertens):

With foldr, $\wedge$I and $\wedge$GE, we can recover para

General elimination works fine
for disjunctions, conjunctions, and the like

But it cannot exploit the structure
of recursive types

Catamorphisms do exactly the same
for disjunctions, conjunctions, and the like

And they can exploit the structure
of recursive types

But they miss some things
general eliminations can do

Paramorphisms combine the two,
giving all the power of both

They are the most promising way
to extend elimination rules to recursive types


Thanks!

Achievement unlocked!
Dependent type theory (50 pts)

Meertens:

"The recursive pattern involved [in paramorphisms] is well known:
it is essentially the same as the standard pattern
used in the so-called elimination rules for a data type
in constructive type theory"

$$\text{para:}\quad \frac{[A] \qquad B \qquad \begin{array}{c} \underbrace{(A), ([A]), (B)} \\ \vdots \\ B \end{array}}{B}$$

$$\frac{x :: [A] \qquad y :: B([\,]) \qquad \begin{array}{c} \underbrace{(a :: A), (l :: [A]), (h :: B(l))} \\ \vdots \\ z(a, l, h) :: B(a : l) \end{array}}{\mathrm{Listelim}(x, y, z) :: B(x)}$$