Adjoint Folds and Unfolds

Or: Scything Through the Thicket of Morphisms

Ralf Hinze

Computing Laboratory, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD, England
ralf.hinze@comlab.ox.ac.uk
http://www.comlab.ox.ac.uk/ralf.hinze/

June 2010

Section 1

Prologue

- Mathematics of Program Construction,

  375th Anniversary of the Groningen University,
  International Conference Groningen, The Netherlands,
  June 26–30 1989. LNCS 375.

  - Mike Spivey, *A categorical approach to the theory of lists*.
  - Grant Malcolm, *Homomorphisms and promotability*.

1.1 Catamorphism

$$f = (\!|b|\!) \quad \Longleftrightarrow \quad f \cdot in = b \cdot \mathsf{F}\, f$$

1.1 Banana-split law

$$(\!|h|\!) \vartriangle (\!|k|\!) \quad = \quad (\!|h \cdot \mathsf{F}\, outl \vartriangle k \cdot \mathsf{F}\, outr|\!)$$

1.1 Proof of banana-split law

$$
\begin{aligned}
& (\!|h|\!) \vartriangle (\!|k|\!)) \cdot in \\
= & \quad \{ \text{ split-fusion } \} \\
& (\!|h|\!) \cdot in \vartriangle (\!|k|\!) \cdot in \\
= & \quad \{ \text{ fold-computation } \} \\
& h \cdot \mathsf{F}\,(\!|h|\!) \vartriangle k \cdot \mathsf{F}\,(\!|k|\!) \\
= & \quad \{ \text{ split-computation } \} \\
& h \cdot \mathsf{F}\,(outl \cdot ((\!|h|\!) \vartriangle (\!|k|\!))) \vartriangle k \cdot \mathsf{F}\,(outr \cdot ((\!|h|\!) \vartriangle (\!|k|\!))) \\
= & \quad \{ \mathsf{F} \text{ functor } \} \\
& h \cdot \mathsf{F}\,outl \cdot \mathsf{F}\,((\!|h|\!) \vartriangle (\!|k|\!)) \vartriangle k \cdot \mathsf{F}\,outr \cdot \mathsf{F}\,((\!|h|\!) \vartriangle (\!|k|\!)) \\
= & \quad \{ \text{ split-fusion } \} \\
& (h \cdot \mathsf{F}\,outl \vartriangle k \cdot \mathsf{F}\,outr) \cdot \mathsf{F}\,((\!|h|\!) \vartriangle (\!|k|\!))
\end{aligned}
$$

1.2 Example: *total*

$$\mathbf{data}\ \mathit{Stack} = \mathit{Empty} \mid \mathit{Push}\,(\mathit{Nat}, \mathit{Stack})$$

$$\begin{aligned}
&\mathit{total} : \mathit{Stack} && \to \mathit{Nat} \\
&\mathit{total}\ \ (\mathit{Empty}) && = 0 \\
&\mathit{total}\ \ (\mathit{Push}\,(n, s)) && = n + \mathit{total}\, s
\end{aligned}$$

1.3 Counterexample: *stack*

$$
\begin{array}{lll}
stack : (Stack, & Stack) & \to Stack \\
stack\ \ (Empty, & bs) & = bs \\
stack\ \ (Push\,(a, as), & bs) & = Push\,(a, stack\,(as, bs))
\end{array}
$$

1.3 Counterexample: *even* and *odd*

$$
\begin{array}{lll}
even : Stack & \to & Bool \\
even\ \ (Empty) & = & True \\
even\ \ (Push\,(0,x)) & = & even\,x \\
even\ \ (Push\,(1,x)) & = & odd\,x
\end{array}
$$

$$
\begin{array}{lll}
odd : Stack & \to & Bool \\
odd\ \ (Empty) & = & False \\
odd\ \ (Push\,(0,x)) & = & odd\,x \\
odd\ \ (Push\,(1,x)) & = & even\,x
\end{array}
$$

1.3 Counterexamples: *fac* and *fib*

$$\mathbf{data}\ Nat = Zero \mid Succ\ Nat$$

$$\begin{aligned}
fac &: Nat \to Nat \\
fac\ &(Zero) = 1 \\
fac\ &(Succ\ n) = Succ\ n \times fac\ n
\end{aligned}$$

$$\begin{aligned}
fib &: Nat \to Nat \\
fib\ &(Zero) = Zero \\
fib\ &(Succ\ Zero) = Succ\ Zero \\
fib\ &(Succ\ (Succ\ n)) = fib\ n + fib\ (Succ\ n)
\end{aligned}$$

1.3 Counterexample: *sum*

$$\mathbf{data}\ \mathsf{List}\ a = \mathit{Nil} \mid \mathit{Cons}\ (a, \mathsf{List}\ a)$$

$$\begin{array}{lll} \mathit{sum} : \mathsf{List}\ \mathit{Nat} & \to \mathit{Nat} \\ \mathit{sum}\ \ (\mathit{Nil}) & = 0 \\ \mathit{sum}\ \ (\mathit{Cons}\ (a, as)) & = a + \mathit{sum}\ as \end{array}$$

1.3 Counterexample: *append*

$$
\begin{array}{lll}
\mathit{append} : \forall\, a\,.\, (\mathsf{List}\, a, & \mathsf{List}\, a) \to \mathsf{List}\, a \\
\mathit{append} \quad (\mathit{Nil}, & \mathit{bs}) \quad = \mathit{bs} \\
\mathit{append} \quad (\mathit{Cons}\,(a, as), & \mathit{bs}) \quad = \mathit{Cons}\,(a, \mathit{append}\,(as, bs))
\end{array}
$$

1.3 Counterexample: *concat*

$$
\begin{array}{lll}
concat : \forall\, a\, .\, \mathsf{List}\,(\mathsf{List}\,a) & \rightarrow \mathsf{List}\,a \\
concat \quad\quad (Nil) & = Nil \\
concat \quad\quad (Cons\,(l, ls)) & = append\,(l, concat\; ls)
\end{array}
$$

1.4 Thicket of morphisms

- Catamorphisms,
- anamorphism,
- paramorphism,
- apomorphisms,
- mutomorphisms,
- zygomorphisms,
- histomorphisms,
- futomorphisms.

Section 2

Mendler-style folds

2.1 Semantics of recursive datatypes

$$\mathbf{data}\ \mathit{Stack} = \mathit{Empty} \mid \mathit{Push}\,(\mathit{Nat}, \mathit{Stack})$$

2.1 Two-level types

Abstracting away from the recursive call.

$$\textbf{data}\ \mathsf{Stack}\ \mathit{stack} = \mathsf{Empty} \mid \mathsf{Push}\ (\mathit{Nat}, \mathit{stack})$$

$$\begin{aligned}&\textbf{instance}\ \mathit{Functor}\ \mathsf{Stack}\ \textbf{where}\\ &\quad \mathit{fmap}\ f\ (\mathsf{Empty}) &&= \mathsf{Empty}\\ &\quad \mathit{fmap}\ f\ (\mathsf{Push}\ (n, s)) &&= \mathsf{Push}\ (n, f\ s)\end{aligned}$$

Tying the recursive knot.

$$\textbf{newtype}\ \mu f = \mathit{In}\ \{\mathit{in}^{\circ} : f\ (\mu f)\}$$

$$\textbf{type}\ \mathit{Stack} = \mu\mathsf{Stack}$$

2.2 Semantics of recursive functions

$$
\begin{aligned}
&\mathit{total} : \mu\mathsf{Stack} && \to \mathit{Nat} \\
&\mathit{total} \quad (\mathit{In}\,(\mathsf{Empty})) && = 0 \\
&\mathit{total} \quad (\mathit{In}\,(\mathsf{Push}\,(n, s))) && = n + \mathit{total}\, s
\end{aligned}
$$

2.2 Abstracting away from the recursive call

$$
\begin{array}{lll}
\mathsf{total} : (\mu\mathsf{Stack} \to \mathit{Nat}) \to (\mu\mathsf{Stack} & \to \mathit{Nat}) \\
\mathsf{total}\ \ \mathit{total}\ \ (\mathit{In}\,(\mathsf{Empty})) & = 0 \\
\mathsf{total}\ \ \mathit{total}\ \ (\mathit{In}\,(\mathsf{Push}\,(n, s))) & = n + \mathit{total}\, s
\end{array}
$$

A function of this type has many fixed points.

2.2 . . . *and* removing *In*

Abstracting away from the recursive call *and* removing *In*.

$$
\begin{array}{llll}
\mathsf{total} : \forall\, x\,.\, (x \to \mathit{Nat}) \to (\mathsf{Stack}\, x & & & \to \mathit{Nat}) \\
\mathsf{total} & \mathit{total} & (\mathsf{Empty}) & = 0 \\
\mathsf{total} & \mathit{total} & (\mathsf{Push}\, (n, s)) & = n + \mathit{total}\, s
\end{array}
$$

A function of this type has a unique fixed point.

Tying the recursive knot.

$$
\begin{array}{lll}
\mathit{total} : \mu\mathsf{Stack} \to \mathit{Nat} & & \\
\mathit{total} & (\mathit{In}\, l) & = \mathsf{total}\, \mathit{total}\, l
\end{array}
$$

2.3 Initial fixed-point equations

An *initial fixed-point equation* in the unknown $x : \mathbb{C}(\mu\mathsf{F}, A)$ has the syntactic form

$$x \cdot in \quad = \quad \Psi\, x \ ,$$

where the base function Ψ has type

$$\Psi : \forall\, X \,.\, \mathbb{C}(X, A) \to \mathbb{C}(\mathsf{F}\, X, A) \ .$$

The naturality of Ψ ensures that x is uniquely defined.

2.3 Mendler-style folds

$$x = (\!|\Psi|\!)_{\mathsf{Id}} \iff x \cdot in = \Psi\, x$$

2.3 Proof of uniqueness

$$\phi : \mathbb{C}(\mathsf{F}\,A, A) \cong (\forall\, X \,.\, \mathbb{C}(X, A) \to \mathbb{C}(\mathsf{F}\,X, A))$$

$$
\begin{array}{ll}
 & x \cdot in = \Psi\, x \\
\iff & \quad \{ \text{ isomorphism } \} \\
 & x \cdot in = \phi\, (\phi^\circ\, \Psi)\, x \\
\iff & \quad \{ \text{ definition of } \phi^\circ, \text{ that is, } \phi^\circ\, \Psi = \Psi\, id \ \} \\
 & x \cdot in = \phi\, (\Psi\, id)\, x \\
\iff & \quad \{ \text{ definition of } \phi, \text{ that is, } \phi\, f = \lambda\, \kappa \,.\, f \cdot \mathsf{F}\, \kappa \ \} \\
 & x \cdot in = \Psi\, id \cdot \mathsf{F}\, x \\
\iff & \quad \{ \text{ initial algebras } \} \\
 & x = (\!|\Psi\, id|\!)
\end{array}
$$

Section 3

Adjoint folds

3.1 Recall *stack*

$$
\begin{array}{lll}
stack : (\mu\mathsf{Stack}, & Stack) \to Stack \\
stack\ \ (In\,(\mathsf{Empty}), & bs) = bs \\
stack\ \ (In\,(\mathsf{Push}\,(a, as)), & bs) = In\,(\mathsf{Push}\,(a, stack\,(as, bs)))
\end{array}
$$

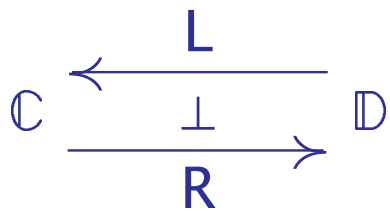
3.2 Adjoint fixed-point equations

Idea: model the context by a functor.

$$x \cdot \mathsf{L}\, in = \Psi\, x$$

Requirement: the functor has to be left adjoint: $\mathsf{L} \dashv \mathsf{R}$.

3.2 Adjunction



$$\phi : \forall\, A\, B\, .\, \mathbb{C}(\mathsf{L}\, A, B) \cong \mathbb{D}(A, \mathsf{R}\, B)$$

3.2 Adjoint initial fixed-point equations

An *adjoint initial fixed-point equation* in the unknown $x : \mathbb{C}(\mathsf{L}\,(\mu\mathsf{F}), A)$ has the syntactic form

$$x \cdot \mathsf{L}\,in \quad = \quad \Psi\,x \ ,$$

where the base function Ψ has type

$$\Psi : \forall\, X : \mathbb{D} \,.\, \mathbb{C}(\mathsf{L}\,X, A) \to \mathbb{C}(\mathsf{L}\,(\mathsf{F}\,X), A) \ .$$

The unique solution is called an *adjoint fold*.
Furthermore, $\phi\,x$ is called the *transposed fold*.

3.2 Adjoint folds

$$x = (\!|\Psi|\!)_{\mathsf{L}} \quad \Longleftrightarrow \quad x \cdot \mathsf{L}\, in = \Psi\, x$$

3.2 Proof of uniqueness

$$
\begin{array}{ll}
 & x \cdot \mathsf{L}\, in = \Psi\, x \\
\Longleftrightarrow & \quad \{ \text{ adjunction } \} \\
 & \phi\, (x \cdot \mathsf{L}\, in) = \phi\, (\Psi\, x) \\
\Longleftrightarrow & \quad \{ \text{ naturality of } \phi, \text{ that is, } \phi\, f \cdot h = \phi\, (f \cdot \mathsf{L}\, h) \} \\
 & \phi\, x \cdot in = \phi\, (\Psi\, x) \\
\Longleftrightarrow & \quad \{ \text{ adjunction } \} \\
 & \phi\, x \cdot in = (\phi \cdot \Psi \cdot \phi^\circ)\, (\phi\, x) \\
\Longleftrightarrow & \quad \{ \text{ universal property of Mendler-style folds } \} \\
 & \phi\, x = (\!|\phi \cdot \Psi \cdot \phi^\circ|\!)_{\mathsf{Id}} \\
\Longleftrightarrow & \quad \{ \text{ adjunction } \} \\
 & x = \phi^\circ\, (\!|\phi \cdot \Psi \cdot \phi^\circ|\!)_{\mathsf{Id}}
\end{array}
$$

3.3 Banana-split law

$$(\!|\Phi|\!)_{\mathsf{L}} \vartriangle (\!|\Psi|\!)_{\mathsf{L}} \quad = \quad (\!|\lambda x \,.\, \Phi\,(outl \cdot x) \vartriangle \Psi\,(outr \cdot x)|\!)_{\mathsf{L}}$$

3.3 Proof of banana-split law

$$
\begin{aligned}
& (\!|\Phi|\!)_{\mathsf{L}} \vartriangle (\!|\Psi|\!)_{\mathsf{L}}) \cdot \mathsf{L}\, in \\
= & \quad \{ \text{ split-fusion } \} \\
& (\!|\Phi|\!)_{\mathsf{L}} \cdot \mathsf{L}\, in \vartriangle (\!|\Psi|\!)_{\mathsf{L}} \cdot \mathsf{L}\, in \\
= & \quad \{ \text{ fold-computation } \} \\
& \Phi\, (\!|\Phi|\!)_{\mathsf{L}} \vartriangle \Psi\, (\!|\Psi|\!)_{\mathsf{L}} \\
= & \quad \{ \text{ split-computation } \} \\
& \Phi\, (outl \cdot ((\!|\Phi|\!)_{\mathsf{L}} \vartriangle (\!|\Psi|\!)_{\mathsf{L}})) \vartriangle \Psi\, (outl \cdot ((\!|\Phi|\!)_{\mathsf{L}} \vartriangle (\!|\Psi|\!)_{\mathsf{L}}))
\end{aligned}
$$

Section 4

Adjunctions

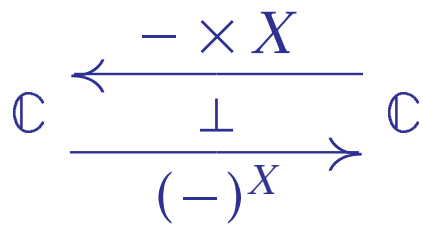
4.1 Recall *stack*

$$
\begin{array}{lll}
\mathit{stack} : \mathsf{L}\,(\mu\mathsf{Stack}) & & \to \mathit{Stack} \\
\mathit{stack} \quad (\mathit{In}\,(\mathsf{Empty}), & \mathit{bs}) & = \mathit{bs} \\
\mathit{stack} \quad (\mathit{In}\,(\mathsf{Push}\,(a, \mathit{as})), & \mathit{bs}) & = \mathit{In}\,(\mathsf{Push}\,(a, \mathit{stack}\,(\mathit{as}, \mathit{bs})))
\end{array}
$$

The type $\mu\mathsf{Stack}$ is embedded in a context L:

$$\mathsf{L}\,A = A \times \mathit{Stack}.$$

4.1 Currying



$$\phi : \forall\, A\, B\, .\, \mathbb{C}(A \times X, B) \cong \mathbb{C}(A, B^X)$$

4.1 Specialising adjoint equations

$$
\begin{array}{ll}
 & x \cdot \mathsf{L}\, in = \Psi\, x \\
\iff & \quad \{ \text{ definition of } \mathsf{L} \ \} \\
 & x \cdot (in \times id) = \Psi\, x \\
\iff & \quad \{ \text{ pointwise } \} \\
 & x\,(in\, a, c) = \Psi\, x\,(a, c)
\end{array}
$$

4.1 *stack* as an adjoint fold

$$
\begin{array}{lll}
\mathsf{stack} : \forall\, x\,.\, (\mathsf{L}\, x \to \mathit{Stack}) \to (\mathsf{L}\, (\mathsf{Stack}\, x) & \to \mathit{Stack}) \\
\mathsf{stack} \quad \mathit{stack} \quad (\mathsf{Empty}, & \mathit{bs}) = \mathit{bs} \\
\mathsf{stack} \quad \mathit{stack} \quad (\mathsf{Push}\, (a, as), & \mathit{bs}) = \\
\quad \mathit{In}\, (\mathsf{Push}\, (a, \mathit{stack}\, (as, bs)))
\end{array}
$$

$$
\begin{array}{lll}
\mathit{stack} : \mathsf{L}\, (\mu\mathsf{Stack}) \to \mathit{Stack} \\
\mathit{stack} \quad (\mathit{In}\, as, bs) = \mathsf{stack}\, \mathit{stack}\, (as, bs)
\end{array}
$$

4.1 The transpose of *stack*

$$\mathsf{R}\,A = A^{Stack}$$

The transposed fold is the curried variant of *stack*.

$$\begin{aligned}
&stack : \mu\mathsf{Stack} && \to \mathsf{R}\,Stack \\
&stack\ \ (In\,\mathsf{Empty}) && = \lambda bs \to bs \\
&stack\ \ (In\,(\mathsf{Push}\,(a, as))) && = \lambda bs \to In\,(\mathsf{Push}\,(a, stack\,as\,bs))
\end{aligned}$$

4.2 Recall *even* and *odd*

$$
\begin{array}{lll}
even : \mu\mathsf{Stack} & \to Bool \\
even\ \ (In\,\mathsf{Empty}) & = True \\
even\ \ (In\,(\mathsf{Push}\,(0,x))) & = even\,x \\
even\ \ (In\,(\mathsf{Push}\,(1,x))) & = odd\,x
\end{array}
$$

$$
\begin{array}{lll}
odd : \mu\mathsf{Stack} & \to Bool \\
odd\ \ (In\,\mathsf{Empty}) & = False \\
odd\ \ (In\,(\mathsf{Push}\,(0,x))) & = odd\,x \\
odd\ \ (In\,(\mathsf{Push}\,(1,x))) & = even\,x
\end{array}
$$

4.2 Speaking categorically

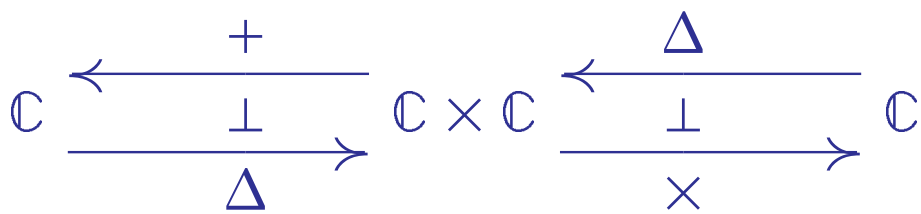
$$evenOdd : \Delta(\mu\mathsf{Stack}) \to \langle Bool, Bool \rangle$$

The type $\mu\mathsf{Stack}$ is embedded in a context Δ:

$$\begin{aligned} \Delta &: \mathbb{C} \to \mathbb{C} \times \mathbb{C} \\ \Delta A &= \langle A, A \rangle \\ \Delta f &= \langle f, f \rangle. \end{aligned}$$

4.2 Adjoints of the diagonal functor

$$\phi : \forall\, A\, B\, .\, \mathbb{C}((+)\, A, B) \cong (\mathbb{C} \times \mathbb{C})(A, \Delta B)$$



$$\phi : \forall\, A\, B\, .\, (\mathbb{C} \times \mathbb{C})(\Delta A, B) \cong \mathbb{C}(A, (\times)\, B)$$

4.2 Specialising adjoint equations

$$x \cdot \Delta in = \Psi\, x$$

$$\Longleftrightarrow$$

$$x_1 \cdot in = \Psi_1\, \langle x_1, x_2 \rangle \quad \text{and} \quad x_2 \cdot in = \Psi_2\, \langle x_1, x_2 \rangle$$

Here, $x_1 = \mathsf{Outl}\, x$, $x_2 = \mathsf{Outr}\, x$, $\Psi_1 = \mathsf{Outl} \cdot \Psi$ and $\Psi_2 = \mathsf{Outr} \cdot \Psi$.

4.2 A special case: paramorphisms

$$
\begin{aligned}
&fac : \mu\mathsf{Nat} && \to Nat \\
&fac\ \ (In\,(\mathsf{Zero})) && = 1 \\
&fac\ \ (In\,(\mathsf{Succ}\,n)) && = In\,(\mathsf{Succ}\,(id\,n)) \times fac\,n
\end{aligned}
$$

$$
\begin{aligned}
&id : \mu\mathsf{Nat} && \to Nat \\
&id\ \ (In\,(\mathsf{Zero})) && = In\,\mathsf{Zero} \\
&id\ \ (In\,(\mathsf{Succ}\,n)) && = In\,(\mathsf{Succ}\,(id\,n))
\end{aligned}
$$

4.2 A special case: histomorphisms

$$
\begin{aligned}
fib :\ & \mu\mathsf{Nat} && \to Nat \\
fib\ & (In\,(\mathsf{Zero})) && = 0 \\
fib\ & (In\,(\mathsf{Succ}\,n)) && = fib'\ n
\end{aligned}
$$

$$
\begin{aligned}
fib' :\ & \mu\mathsf{Nat} && \to Nat \\
fib'\ & (In\,(\mathsf{Zero})) && = 1 \\
fib'\ & (In\,(\mathsf{Succ}\,n)) && = fib\,n + fib'\ n
\end{aligned}
$$

adjunction	initial fixed-point equation	final fixed-point equation
$\mathsf{L} \dashv \mathsf{R}$	$x \cdot \mathsf{L}\, in = \Psi\, x$ $\phi\, x \cdot in = (\phi \cdot \Psi \cdot \phi^\circ)\, (\phi\, x)$	$\mathsf{R}\, out \cdot x = \Psi\, x$ $out \cdot \phi^\circ\, x = (\phi^\circ \cdot \Psi \cdot \phi)\, (\phi^\circ\, x)$
$\mathsf{Id} \dashv \mathsf{Id}$	standard fold standard fold	standard unfold standard unfold
$(- \times X) \dashv (-^X)$	parametrised fold fold to an exponential	curried unfold unfold from a pair
$(+) \dashv \Delta$	recursion from a coproduct of mutually recursive types mutual value recursion on mutually recursive types	mutual value recursion single recursion from a coproduct domain
$\Delta \dashv (\times)$	mutual value recursion single recursion to a product domain	recursion to a product of mutually recursive types mutual value recursion on mutually recursive types
$\mathsf{Lsh}_X \dashv (- X)$	—	monomorphic unfold unfold from a left shift
$(- X) \dashv \mathsf{Rsh}_X$	monomorphic fold fold to a right shift	—
$\mathsf{Lan}_\mathsf{J} \dashv (- \circ \mathsf{J})$	—	polymorphic unfold unfold from a left Kan extension
$(- \circ \mathsf{J}) \dashv \mathsf{Ran}_\mathsf{J}$	polymorphic fold fold to a right Kan extension	—

Section 5

Epilogue

5.1 Summary and related work

Summary:

- Adjoint (un-) folds capture many recursion schemes:
  - catamorphisms and anamorphism,
  - folds with parameter,
  - paramorphism and apomorphisms,
  - mutomorphisms,
  - zygomorphisms,
  - (histomorphisms and futomorphisms).
- Adjunctions play a central role.

Related work:

- R. Bird, R. Paterson: “Generalised folds for nested datatypes”.

5.2 Limitations

- Simultaneous recursion does not fit under the umbrella.

$$
\begin{array}{lll}
zip : (\mathsf{List}\, a, & \mathsf{List}\, b) & \to \mathsf{List}\,(a, b) \\
zip\ \ (Nil, & bs) & = Nil \\
zip\ \ (as, & Nil) & = Nil \\
zip\ \ (Cons\,(a, as), & Cons\,(b, bs)) & = Cons\,((a, b), zip\,(as, bs))
\end{array}
$$

- However, one can establish

$$x = (\!|\Psi|\!)_{\times} \iff x \cdot (\times)\ in = \Psi\, x$$

using a different technique: colimits. See, R. Bird, R. Paterson: “Generalised folds for nested datatypes”.