Generic Programming with Adjunctions

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/

March 2010

Part 1

Prologue

1.0 Outline

1.1 What?

Haskell programmers have embraced

- functors,
- natural transformations,
- initial algebras,
- final coalgebras,
- monads,
- . . .

It is time to turn our attention to adjunctions.

1.2 Catamorphism

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

1.2 Banana-split law

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

1.2 Proof of banana-split law

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

1.2 Example: *total*

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

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

1.2 Two-level types

Abstracting away from the recursive call.

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

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

Tying the recursive knot.

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

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

1.2 Two-level functions

Structure.

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

Tying the recursive knot.

$$\begin{aligned} &\mathit{total} : \mu\mathsf{Stack} \to \mathit{Nat} \\ &\mathit{total}\ \ (\mathit{In}\, s) \ \ = \mathsf{total}\,(\mathit{fmap}\, \mathit{total}\, s) \end{aligned}$$

1.2 Counterexample: *stack*

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

1.2 Counterexamples: *fac* and *fib*

$$\textbf{data}\ Nat = Z \mid S\ Nat$$

$$\begin{aligned}
&fac : Nat \to Nat\\
&fac\ (Z) = 1\\
&fac\ (S\ n) = S\ n * fac\ n
\end{aligned}$$

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

1.2 Counterexample: *sum*

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

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

1.2 Counterexample: *append*

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

1.2 Counterexample: *concat*

$$
\begin{aligned}
&\mathit{concat} : \forall\, a\,.\ \mathsf{List}\,(\mathsf{List}\,a) &&\to \mathsf{List}\,a\\
&\mathit{concat}\quad (\mathit{Nil}) &&= \mathit{Nil}\\
&\mathit{concat}\quad (\mathit{Cons}\,(l, \mathit{ls})) &&= \mathit{append}\,(l, \mathit{concat}\ \mathit{ls})
\end{aligned}
$$

1.3 References

The lectures are based on:

- Part 1: R. Hinze: A category theory primer, SSGIP 2010.
- Part 2 & 3: R. Hinze: Adjoint Folds and Unfolds, MPC'10.
- Part 4: R. Hinze: Type Fusion.

Further reading:

- S. Mac Lane: Categories for the Working Mathematician.
- M. Fokkinga, L. Meertens: Adjunctions.
- R. Bird, R. Paterson: Generalised folds for nested datatypes.

1.4 Overview

- Part 0: Prologue
- Part 1: Category theory
- Part 2: Adjoint folds and unfolds
- Part 3: Adjunctions
- Part 4: Application: Type fusion
- Part 5: Epilogue

Part 2

Category theory

2.0 Outline

2.1 Category

- *objects:* $A \in \mathbb{C}$,
- *arrows:* $f \in \mathbb{C}(A, B)$,
- *identity:* $id_A \in \mathbb{C}(A, A)$,
- *composition:* if $f \in \mathbb{C}(A, B)$ and $g \in \mathbb{C}(B, C)$, then $g \cdot f \in \mathbb{C}(A, C)$,
- $\cdot$ is associative with *id* as its neutral element.

2.1 Example: a preorder P

- *objects:* $a \in P$,
- *arrows:* $a \leqslant b$,
- *identity:* $a \leqslant a$ (reflexivity),
- *composition:* if $a \leqslant b$ and $b \leqslant c$, then $a \leqslant c$ (transitivity).

NB There is at most one arrow between two objects.

2.1 Example: Set

- *objects:* sets,
- *arrows:* total functions,
- *identity:* $id\, x = x$,
- *composition:* function composition $(g \cdot f)\, x = g\,(f\, x)$.

2.1 Example: Mon

- *objects:* monoids $\langle A, \epsilon, \mathbin{+\!\!+} \rangle$,
- *arrows:* monoid homomorphisms
  $h : \langle A, 0, + \rangle \to \langle B, 1, * \rangle$:
  $$\begin{aligned} h\,0 &= 1 \\ h\,(x + y) &= h\,x * h\,y, \end{aligned}$$
- *identity:* $id\,x = x$,
- *composition:* function composition $(g \cdot f)\,x = g\,(f\,x)$.

2.1 Functor

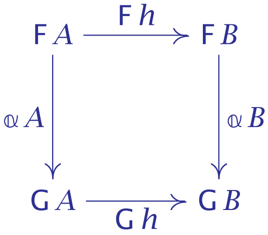
- $\mathsf{F} : \mathbb{C} \to \mathbb{D}$,
- action on objects,
- action on arrows,
- if $f \in \mathbb{C}(A, B)$, then $\mathsf{F}\, f \in \mathbb{D}(\mathsf{F}\, A, \mathsf{F}\, B)$
- $\mathsf{F}\, id_A = id_{\mathsf{F}\, A}$,
- $\mathsf{F}\,(g \cdot f) = \mathsf{F}\, g \cdot \mathsf{F}\, f$.

2.1 Example: the forgetful functor

- $\mathsf{U} : \mathbf{Mon} \to \mathbf{Set}$,
- action on objects: $\mathsf{U}\,\langle A, \epsilon, +\!\!+ \rangle = A$,
- action on arrows: $\mathsf{U}\,f = f$.

2.1 Natural transformation

- let $\mathsf{F}, \mathsf{G} : \mathbb{C} \to \mathbb{D}$ be two parallel functors,
- a transformation $\alpha : \mathsf{F} \to \mathsf{G}$ is a collection of arrows: for each object $A \in \mathbb{C}$ there is an arrow $\alpha\, A \in \mathbb{D}(\mathsf{F}\, A, \mathsf{G}\, A)$,
- a natural transformation $\alpha : \mathsf{F} \dot{\to} \mathsf{G}$ additionally satisfies $\mathsf{G}\, h \cdot \alpha\, A = \alpha\, B \cdot \mathsf{F}\, h$ for all arrows $h \in \mathbb{C}(A, B)$.



2.2 Cat

- *objects:* small categories,
- *arrows:* functors,
- *identity:* identity functor: $\mathsf{Id}_{\mathbb{C}}\, A = A$ and $\mathsf{Id}_{\mathbb{C}}\, f = f$,
- *composition:* $(\mathsf{G} \circ \mathsf{F})\, A = \mathsf{G}\,(\mathsf{F}\, A)$ and $(\mathsf{G} \circ \mathsf{F})\, f = \mathsf{G}\,(\mathsf{F}\, f)$.

2.2 Functor category $\mathbb{D}^{\mathbb{C}}$

- let $\mathbb{C}$ and $\mathbb{D}$ be two categories,
- *objects:* functors $\mathbb{C} \to \mathbb{D}$,
- *arrows:* natural transformations $\mathsf{F} \dot{\to} \mathsf{G}$,
- *identity:* $id_{\mathsf{F}}\, A = id_{\mathsf{F}\, A}$,
- *composition:* $(\beta \cdot \alpha)\, A = \beta\, A \cdot \alpha\, A$.

2.2 Opposite category $\mathbb{C}^{\mathsf{op}}$

- let $\mathbb{C}$ be a category,
- *objects:* $A \in \mathbb{C}^{\mathsf{op}}$ if $A \in \mathbb{C}$
- *arrows:* $f \in \mathbb{C}^{\mathsf{op}}(A, B)$ if $f \in \mathbb{C}(B, A)$
- *identity:* $id_A \in \mathbb{C}(A, A)$,
- *composition:* $g \cdot f \in \mathbb{C}^{\mathsf{op}}(A, C)$ if $f \cdot g \in \mathbb{C}(C, A)$.

2.2 Product category $\mathbb{C}_1 \times \mathbb{C}_2$

- let $\mathbb{C}_1$ and $\mathbb{C}_2$ be two categories,
- *objects:* $\langle A_1, A_2 \rangle \in \mathbb{C}_1 \times \mathbb{C}_2$ if $A_1 \in \mathbb{C}_1$ and $A_2 \in \mathbb{C}_2$,
- *arrows:* $\langle f_1, f_2 \rangle \in (\mathbb{C}_1 \times \mathbb{C}_2)(\langle A_1, A_2 \rangle, \langle B_1, B_2 \rangle)$ if $f_1 \in \mathbb{C}_1(A_1, B_1)$ and $f_2 \in \mathbb{C}_2(A_2, B_2)$,
- *identity:* $id = \langle id, id \rangle$,
- *composition:* $\langle g_1, g_2 \rangle \cdot \langle f_1, f_2 \rangle = \langle g_1 \cdot f_1, g_2 \cdot f_2 \rangle$.

2.2 Outl, Outr and Δ

- projection functors:

$$\begin{aligned} \mathsf{Outl} &: \mathbb{C} \times \mathbb{D} \to \mathbb{C}; \\ \mathsf{Outl}\,\langle A, B\rangle &= A; \\ \mathsf{Outl}\,\langle f, g\rangle &= f; \end{aligned} \qquad \begin{aligned} \mathsf{Outr} &: \mathbb{C} \times \mathbb{D} \to \mathbb{D}; \\ \mathsf{Outr}\,\langle A, B\rangle &= B; \\ \mathsf{Outr}\,\langle f, g\rangle &= g. \end{aligned}$$

- diagonal functor:

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

2.2 The hom-functor

- $\mathbb{C}(-, =) : \mathbb{C}^{\mathsf{op}} \times \mathbb{C} \to \mathbf{Set}$,
- action on objects: $\mathbb{C}(-, =)\,\langle A, B\rangle = \mathbb{C}(A, B)$,
- action on arrows: $\mathbb{C}(-, =)\,\langle f, g\rangle = \lambda\, h \,.\, g \cdot h \cdot f$,
- shorthand: $\mathbb{C}(f, g)\, h = g \cdot h \cdot f$.

2.3 Initial object

The object 0 is initial if for each object $B \in \mathbb{C}$ there is exactly one arrow from 0 to B, denoted $¡_B$ (pronounce "gnab").

$$0 \overset{¡_B}{\dashrightarrow} B$$

2.3 Final object

Dually, 1 is a final object if for each object $A \in \mathbb{C}$ there is a unique arrow from A to 1, denoted $!_A$ (pronounce “bang”).

$$A \overset{!_A}{\dashrightarrow} 1$$

2.4 Product

The *product* of two objects B_1 and B_2 consists of

- an object written $B_1 \times B_2$,
- a pair of arrows $outl : B_1 \times B_2 \to B_1$ and $outr : B_1 \times B_2 \to B_2$,

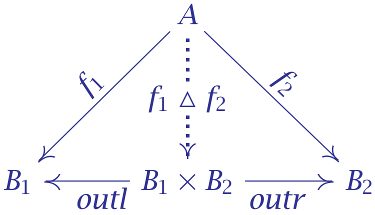
and satisfies the following *universal property*:

- for each object A,
- for each pair of arrows $f_1 : A \to B_1$ and $f_2 : A \to B_2$,
- there exists an arrow $f_1 \vartriangle f_2 : A \to B_1 \times B_2$ such that

$$f_1 = outl \cdot g \;\wedge\; f_2 = outr \cdot g \iff f_1 \vartriangle f_2 = g,$$

for all $g : A \to B_1 \times B_2$.

2.4 Product



2.4 Laws

- *computation laws*:

$$f_1 = outl \cdot (f_1 \vartriangle f_2);$$
$$f_2 = outr \cdot (f_1 \vartriangle f_2),$$

- *reflection law*:

$$outl \vartriangle outr = id_{A \times B}.$$

2.4 Laws

- *fusion law*:

$$(f_1 \vartriangle f_2) \cdot h \quad = \quad f_1 \cdot h \vartriangle f_2 \cdot h,$$

- action of $\times$ on arrows:

$$f_1 \times f_2 \quad = \quad f_1 \cdot outl \vartriangle f_2 \cdot outr,$$

- *functor fusion law*:

$$(k_1 \times k_2) \cdot (f_1 \vartriangle f_2) \quad = \quad k_1 \cdot f_1 \vartriangle k_2 \cdot f_2,$$

- *outl* and *outr* are natural transformations:

$$k_1 \cdot outl \quad = \quad outl \cdot (k_1 \times k_2);$$

$$k_2 \cdot outr \quad = \quad outr \cdot (k_1 \times k_2).$$

2.4 Proof of functor fusion

$$
\begin{aligned}
& (k_1 \times k_2) \cdot (f_1 \vartriangle f_2) \\
= \quad & \{ \text{ definition of } \times \} \\
& (k_1 \cdot \mathit{outl} \vartriangle k_2 \cdot \mathit{outr}) \cdot (f_1 \vartriangle f_2) \\
= \quad & \{ \text{ fusion } \} \\
& k_1 \cdot \mathit{outl} \cdot (f_1 \vartriangle f_2) \vartriangle k_2 \cdot \mathit{outr} \cdot (f_1 \vartriangle f_2) \\
= \quad & \{ \text{ computation } \} \\
& k_1 \cdot f_1 \vartriangle k_2 \cdot f_2
\end{aligned}
$$

2.4 Naturality

- fusion and functor fusion:

$$(\vartriangle) \quad : \quad \forall A\, B\,.\ (\mathbb{C} \times \mathbb{C})(\Delta A, B) \to \mathbb{C}(A, \times B),$$

- naturality of *outl* and *outr*:

$$outl \quad : \quad \forall B\,.\ \mathbb{C}(\times B, \mathsf{Outl}\, B);$$

$$outr \quad : \quad \forall B\,.\ \mathbb{C}(\times B, \mathsf{Outr}\, B),$$

or more succinctly

$$\langle outl, outr\rangle \quad : \quad \forall B\,.\ (\mathbb{C} \times \mathbb{C})(\Delta(\times B), B).$$

2.5 Adjunction

$$\mathbb{C} \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\perp}}}} \mathbb{D}$$

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

2.5 Adjoints, adjuncts and units

- left and right adjoints:

$$\begin{aligned} \mathsf{L}\,g &= \phi^{\circ}\,(\eta \cdot g), \\ \mathsf{R}\,f &= \phi\,(f \cdot \epsilon), \end{aligned}$$

- left and right adjuncts:

$$\begin{aligned} \phi^{\circ}\,g &= \epsilon \cdot \mathsf{L}\,g, \\ \phi\,f &= \mathsf{R}\,f \cdot \eta, \end{aligned}$$

- counit and unit:

$$\begin{aligned} \epsilon &= \phi^{\circ}\,\mathit{id}, \\ \eta &= \phi\,\mathit{id}. \end{aligned}$$

2.5 Adjoints of the diagonal functor

$$f = \langle \mathit{outl}, \mathit{outr} \rangle \cdot \Delta g \quad \Longleftrightarrow \quad \vartriangle f = g$$

$$\mathbb{C} \;\underset{\Delta}{\overset{+}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}}\; \mathbb{C} \times \mathbb{C} \;\underset{\times}{\overset{\Delta}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}}\; \mathbb{C}$$

$$f = \triangledown g \quad \Longleftrightarrow \quad \Delta f \cdot \langle \mathit{inl}, \mathit{inr} \rangle = g$$

2.5 Left adjoint of the forgetful functor

$$\mathbf{Mon} \underset{\mathsf{U}}{\overset{\mathsf{List}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathbf{Set}$$

ϕ° *introduction / elimination*	ϕ *elimination / introduction*
Universal property $f = \phi^\circ\, g \iff \phi\, f = g$	
$\epsilon : \mathbb{C}(\mathsf{L}\,(\mathsf{R}\,B), B)$ $\epsilon = \phi^\circ\, id$	$\eta : \mathbb{D}(A, \mathsf{R}\,(\mathsf{L}\,A))$ $\phi\, id = \eta$
— / *computation law* η-rule / β-rule $f = \phi^\circ\,(\phi\, f)$	*computation law* / — β-rule / η-rule $\phi\,(\phi^\circ\, g) = g$
reflection law / — simple η-rule / simple β-rule $id = \phi^\circ\, \eta$	— / *reflection law* simple β-rule / simple η-rule $\phi\, \epsilon = id$

ϕ° introduction / elimination	*ϕ elimination / introduction*
Universal property $f = \phi^\circ\, g \iff \phi\, f = g$	
functor fusion law / — ϕ° is natural in A $\phi^\circ\, g \cdot \mathsf{L}\, h = \phi^\circ\, (g \cdot h)$	*— / fusion law* ϕ is natural in A $\phi\, f \cdot h = \phi\, (f \cdot \mathsf{L}\, h)$
fusion law / — ϕ° is natural in B $k \cdot \phi^\circ\, g = \phi^\circ\, (\mathsf{R}\, k \cdot g)$	*— / functor fusion law* ϕ is natural in B $\mathsf{R}\, k \cdot \phi\, f = \phi\, (k \cdot f)$
ϵ is natural in B $k \cdot \epsilon = \epsilon \cdot \mathsf{L}\, (\mathsf{R}\, k)$	η is natural in A $\mathsf{R}\, (\mathsf{L}\, h) \cdot \eta = \eta \cdot h$

2.6 Yoneda lemma

Let $\mathsf{H} : \mathbb{C} \to \mathbf{Set}$ be a functor, and let $B \in \mathbb{C}$ be an object.

$$\mathsf{H}\,B \;\cong\; \mathbb{C}(B, -) \mathrel{\dot{\to}} \mathsf{H}$$

The functions witnessing the isomorphism are

$$\begin{aligned} \phi\, s &= \lambda\,\kappa\,.\,\mathsf{H}\,\kappa\,s, \\ \phi^{\circ}\,\alpha &= \alpha\,B\,id_B. \end{aligned}$$

NB Continuation-passing style is a special case: $\mathsf{H} = \mathbb{C}(A, -)$.

Part 3

Adjoint folds and unfolds

3.0 Outline

3.1 Example: *total*

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

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

3.1 Fixed-point equations

- both *Stack* and *total* are given by recursion equations,
- meaning of $x = \Psi\, x$?
- a solves the equation iff a is a fixed point of Ψ,
- Ψ is called the base function.

3.1 Two-level types

Abstracting away from the recursive call.

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

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

Tying the recursive knot.

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

$$\mathbf{type}\ Stack = \mu\mathsf{Stack}$$

3.1 Speaking categorically

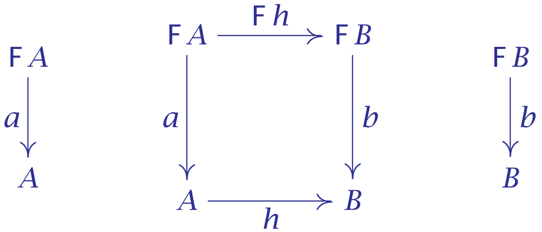
- functor: $\mathsf{Stack}\,A = 1 + \mathit{Nat} \times A$,
- a *Stack*-algebra:

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

- $\mathsf{total} = \mathit{zero} \triangledown \mathit{plus}$,
- *Stack*-algebra: $\langle \mathit{Nat}, \mathsf{total} \rangle$.

3.1 The category of F-algebras Alg(F)

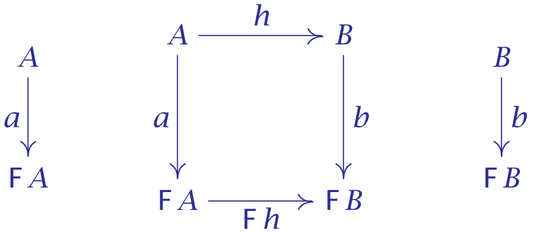
- let $\mathsf{F} : \mathbb{C} \to \mathbb{C}$ be an endofunctor,
- *objects:* $\langle A, a \rangle$ with $A \in \mathbb{C}$ and $a \in \mathbb{C}(\mathsf{F}\,A, A)$,
- *arrows:* F-homomorphisms, $h : \langle A, a \rangle \to \langle B, b \rangle$ if $h \in \mathbb{C}(A, B)$ such that $h \cdot a = b \cdot \mathsf{F}\,h$,



- *identity:* $id_A : \langle A, a \rangle \to \langle A, a \rangle$,
- *composition:* in $\mathbb{C}$.

3.1 The category of F-coalgebras Coalg(F)

- let $\mathsf{F} : \mathbb{C} \to \mathbb{C}$ be an endofunctor,
- *objects:* $\langle A, a \rangle$ with $A \in \mathbb{C}$ and $a \in \mathbb{C}(A, \mathsf{F}\,A)$,
- *arrows:* F-homomorphisms, $h : \langle A, a \rangle \to \langle B, b \rangle$ if $h \in \mathbb{C}(A, B)$ such that $\mathsf{F}\,h \cdot a = b \cdot h$,



- *identity:* $id_A : \langle A, a \rangle \to \langle A, a \rangle$,
- *composition:* in $\mathbb{C}$.

3.1 Fixed points of functors

- initial object in **Alg**(F): *initial* F*-algebra* $\langle \mu\mathsf{F}, in \rangle$,
- $\mu\mathsf{F}$ is the least fixed point of F,
- $in : \mathsf{F}\,(\mu\mathsf{F}) \cong \mu\mathsf{F}$,
- final object in **Coalg**(F): *final* F*-coalgebra* $\langle \nu\mathsf{F}, out \rangle$,
- $\nu\mathsf{F}$ is the greatest fixed point of F,
- $out : \nu\mathsf{F} \cong \mathsf{F}\,(\nu\mathsf{F})$.

3.1 Coq: inductive and coinductive types

Inductive *Nat* : *Type* :=
| *Zero* : *Nat*
| *Succ* : *Nat* → *Nat*.

Inductive *Stack* : *Type* :=
| *Empty* : *Stack*
| *Push* : *Nat* → *Stack* → *Stack*.

CoInductive *Stream* : *Type* :=
| *Cons* : *Nat* → *Stream* → *Stream*.

3.2 Semantics of recursive functions

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

3.2 Abstracting away from the recursive call

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

A function of this type has many fixed points.

3.2 ...removing in

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

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

A function of this type has a unique ‘fixed point’.

Tying the recursive knot.

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

3.2 Example: *from*

$$\mathbf{data}\ \mathsf{Sequ} = \mathit{Next}\ (\mathit{Nat}, \mathsf{Sequ})$$

$$\begin{aligned} &\mathit{from} : \mathit{Nat} \to \mathsf{Sequ} \\ &\mathit{from}\ \ n \quad = \mathit{Next}\ (n, \mathit{from}\ (n + 1)) \end{aligned}$$

3.2 Two-level types and functions

$$\textbf{data}\ \mathsf{Sequ}\ \mathit{sequ} = \mathsf{Next}\ (\mathit{Nat}, \mathit{sequ})$$

$$\begin{aligned}
&\mathsf{from} : \forall x\,.\,(\mathit{Nat} \to x) \to (\mathit{Nat} \to \mathsf{Sequ}\ x)\\
&\mathsf{from}\quad \mathit{from}\quad n \quad = \mathsf{Next}\ (n, \mathit{from}\ (n+1))
\end{aligned}$$

$$\begin{aligned}
&\mathit{from} : \mathit{Nat} \to \nu\mathsf{Sequ}\\
&\mathit{from}\ \ n \quad = \mathit{Out}^{\circ}\ (\mathsf{from}\ \mathit{from}\ n)\ .
\end{aligned}$$

3.2 Initial fixed-point equations

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

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

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 *termination*.

3.2 Guarded by destructors

$$x = \Psi\, x \cdot in^{\circ}$$

$$x \in \mathbb{C}(\mu\mathsf{F}, A)$$

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

$$\mu\mathsf{F} \xrightarrow{in^{\circ}} \mathsf{F}\,(\mu\mathsf{F}) \xrightarrow{\Psi\,x} A$$

3.2 Mendler-style folds

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

3.2 Proof of uniqueness

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

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

3.2 Final fixed-point equations

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

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

where the base function Ψ has type

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

The naturality of Ψ ensures *productivity*.

3.2 Guarded by constructors

$$x \;=\; out^{\circ} \cdot \Psi\, x$$

$$x \in \mathbb{C}\,(A, \nu\mathsf{F})$$

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

$$A \xrightarrow{\;\Psi\, x\;} \mathsf{F}\,(\nu\mathsf{F}) \xrightarrow{\;out^{\circ}\;} \nu\mathsf{F}$$

3.2 Mendler-style unfolds

$$x = [(\Psi)]_{\mathsf{Id}} \iff out \cdot x = \Psi\, x$$

3.2 Mutual type recursion

$$\textbf{data}\ \mathit{Tree} = \mathit{Node}\ \mathit{Nat}\ \mathit{Trees}$$

$$\textbf{data}\ \mathit{Trees} = \mathit{Nil} \mid \mathit{Cons}\,(\mathit{Tree}, \mathit{Trees})$$

$$\begin{aligned}
&\mathit{flattena} : \mathit{Tree} && \to \mathit{Stack} \\
&\mathit{flattena}\ \ (\mathit{Node}\ n\ ts) && = \mathit{Push}\,(n, \mathit{flattens}\ ts)
\end{aligned}$$

$$\begin{aligned}
&\mathit{flattens} : \mathit{Trees} && \to \mathit{Stack} \\
&\mathit{flattens}\ \ (\mathit{Nil}) && = \mathit{Empty} \\
&\mathit{flattens}\ \ (\mathit{Cons}\,(t, ts)) && = \mathit{stack}\,(\mathit{flattena}\ t, \mathit{flattens}\ ts)
\end{aligned}$$

3.2 Speaking categorically

Idea: view *Tree* and *Trees* as a fixed point in a *product category*.

$$\mathsf{T}\,\langle A, B\rangle = \langle Nat \times B, 1 + A \times B\rangle$$

$$flatten \in (\mathbb{C} \times \mathbb{C})(\mu\mathsf{T}, \langle Stack, Stack\rangle)$$

3.2 Specialising fixed-point equations

An equation in $\mathbb{C} \times \mathbb{D}$ corresponds to two equations, one in $\mathbb{C}$ and one in $\mathbb{D}$.

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

$$\iff$$

$$x_1 \cdot in_1 = \Psi_1\, \langle x_1, x_2 \rangle \quad \text{and} \quad x_2 \cdot in_2 = \Psi_2\, \langle x_1, x_2 \rangle$$

Here, $x_1 = \mathsf{Outl}\, x$, $x_2 = \mathsf{Outr}\, x$, $in_1 = \mathsf{Outl}\, in$, $in_2 = \mathsf{Outr}\, in$, $\Psi_1 = \mathsf{Outl} \cdot \Psi$ and $\Psi_2 = \mathsf{Outr} \cdot \Psi$.

3.2 Parametric datatypes

$$\mathbf{data}\ \mathsf{Perfect}\ a = \mathit{Zero}\ a \mid \mathit{Succ}\ (\mathsf{Perfect}\ (a, a))$$

$$\begin{aligned}
&\mathit{size} : \forall\, a\,.\ \mathsf{Perfect}\ a \to \mathit{Nat}\\
&\mathit{size}\quad (\mathit{Zero}\ a) \ = 1\\
&\mathit{size}\quad (\mathit{Succ}\ p) \ = 2 * \mathit{size}\ p
\end{aligned}$$

3.2 Speaking categorically

Idea: view Perfect as a fixed point in a *functor category*.

$$\mathsf{P}\,\mathsf{F} = \Lambda A\,.\,A + \mathsf{F}\,(A \times A)$$

The second-order functor F sends a functor to a functor.

$$size : \mu\mathsf{P} \mathrel{\dot{\to}} \mathsf{K}\,Nat$$

NB $\mathsf{K} : \mathbb{D} \to \mathbb{D}^{\mathbb{C}}$ is the constant functor $\mathsf{K}\,A = \Lambda B\,.\,A$.

3.2 Specialising fixed-point equations

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

$$\iff$$

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

NB Type application and abstraction are invisible in Haskell.

	initial fixed-point equation $x \cdot in = \Psi\, x$	final fixed-point equation $out \cdot x = \Psi\, x$
Set	inductive type standard fold	coinductive type standard unfold
Cpo	—	continuous coalgebra continuous unfold
$\mathbf{Cpo}_{\perp}$	continuous algebra strict continuous fold	continuous coalgebra strict continuous unfold
	$(\mu\mathsf{F} \cong \nu\mathsf{F})$	
$\mathbb{C} \times \mathbb{D}$	mutually rec. ind. types mutually rec. folds	mutually rec. coind. types mutually rec. unfolds
$\mathbb{D}^{\mathbb{C}}$	inductive type functor higher-order fold	coinductive type functor higher-order unfold

3.3 Counterexample: *stack*

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

3.3 Counterexample: *nats* and *squares*

$$
\begin{aligned}
&nats : Nat \to \nu\mathsf{Sequ}\\
&nats\ \ n = Out^{\circ}\,(\mathsf{Next}\,(n, squares\,n))
\end{aligned}
$$

$$
\begin{aligned}
&squares : Nat \to \nu\mathsf{Sequ}\\
&squares\ \ n = Out^{\circ}\,(\mathsf{Next}\,(n * n, nats\,(n+1)))
\end{aligned}
$$

3.3 Adjoint fixed-point equations

Idea: model the context by a functor.

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

Requirement: the functors have to be adjoint: $\mathsf{L} \dashv \mathsf{R}$.

3.3 Adjoint initial fixed-point equations

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

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

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*.

3.3 Proof of uniqueness

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

3.3 Adjoint folds

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

3.3 Banana-split law

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

3.3 Proof of banana-split law

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

3.3 Adjoint final fixed-point equations

An *adjoint final fixed-point equation* in the unknown $x \in \mathbb{D}(A, \mathsf{R}\,(\nu \mathsf{F}))$ has the syntactic form

$$\mathsf{R}\,\mathit{out} \cdot x \quad = \quad \Psi\, x \;\;,$$

where the base function Ψ has type

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

The unique solution is called an *adjoint unfold*.

3.3 Adjoint unfolds

$$x = [(\Psi)]_{\mathsf{R}} \iff \mathsf{R}\, out \cdot x = \Psi\, x$$

Part 4

Adjunctions

4.0 Outline

4.1 Recall: Adjoint fixed-point equations

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

Requirement: the functors have to be adjoint: $\mathsf{L} \dashv \mathsf{R}$.

4.1 Identity

$$\mathbb{C} \underset{\mathsf{Id}}{\overset{\mathsf{Id}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathbb{C}$$

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

Adjoint fixed-point equations subsume Mendler-style ones.

4.2 Recall: *stack*

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

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

$$
\begin{aligned}
\mathsf{L}\,A &= A \times \mathit{Stack} \\
\mathsf{L}\,f &= f \times \mathit{id}_{\mathit{Stack}}.
\end{aligned}
$$

4.2 Currying

$$\mathbb{C} \underset{(-)^X}{\overset{- \times X}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathbb{C}$$

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

4.2 Specialising adjoint equations

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

$$
\begin{aligned}
& \mathsf{R}\, out \cdot x = \Psi\, x \\
\Longleftrightarrow \quad & \{ \text{ definition of } \mathsf{R} \ \} \\
& (out\, \cdot) \cdot x = \Psi\, x \\
\Longleftrightarrow \quad & \{ \text{ pointwise } \} \\
& out\,(x\, a\, c) = \Psi\, x\, a\, c
\end{aligned}
$$

4.2 *stack* as an adjoint fold

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

$$
\begin{aligned}
&\mathit{stack} : \mathsf{L}\,(\mu\mathsf{Stack}) \to \mathit{Stack}\\
&\mathit{stack}\;(\mathit{In}\,\mathit{as}, \mathit{bs}) = \mathsf{stack}\;\mathit{stack}\;(\mathit{as}, \mathit{bs})
\end{aligned}
$$

4.2 The transpose of *stack*

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

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

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

4.2 Recall: *append*

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

4.2 Two-level types

$$\textbf{data}\ \mathsf{LIST}\ \mathit{list}\ a = \mathsf{Nil} \mid \mathsf{Cons}\ (a, \mathit{list}\ a)$$

$$\begin{aligned}
&\textbf{instance}\ (\mathit{Functor}\ \mathit{list}) \Rightarrow \mathit{Functor}\ (\mathsf{LIST}\ \mathit{list})\ \textbf{where} \\
&\quad \mathit{fmap}\ f\ (\mathsf{Nil}) = \mathsf{Nil} \\
&\quad \mathit{fmap}\ f\ (\mathsf{Cons}\ (a, \mathit{as})) = \mathsf{Cons}\ (f\ a, \mathit{fmap}\ f\ \mathit{as})
\end{aligned}$$

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

4.2 *append* as a natural transformation

Defining $(\mathsf{F} \mathbin{\dot{\times}} \mathsf{G})\, A = \mathsf{F}\, A \times \mathsf{G}\, A$, we can view *append* as a natural transformation:

$$append : \mathsf{List} \mathbin{\dot{\times}} \mathsf{List} \mathbin{\dot{\to}} \mathsf{List}.$$

We have to find the right adjoint of the lifted product $- \mathbin{\dot{\times}} \mathsf{H}$.

4.2 Deriving the right adjoint

$$
\begin{aligned}
& \mathsf{G}^{\mathsf{H}} A \\
\cong\ & \quad \{ \text{ Yoneda lemma } \} \\
& \mathbb{C}(A, -) \stackrel{.}{\to} \mathsf{G}^{\mathsf{H}} \\
\cong\ & \quad \{ \text{ requirement: } - \dot{\times} \mathsf{H} \dashv -^{\mathsf{H}} \} \\
& \mathbb{C}(A, -) \dot{\times} \mathsf{H} \stackrel{.}{\to} \mathsf{G} \\
\cong\ & \quad \{ \text{ natural transformation } \} \\
& \forall X : \mathbb{C} \,.\, \mathbb{C}(A, X) \times \mathsf{H}\, X \to \mathsf{G}\, X \\
\cong\ & \quad \{ - \times X \dashv -^{X} \} \\
& \forall X : \mathbb{C} \,.\, \mathbb{C}(A, X) \to (\mathsf{G}\, X)^{\mathsf{H}\, X} \ .
\end{aligned}
$$

NB We assume that the functor category is $\mathbf{Set}^{\mathbb{C}}$ so $\mathsf{G}^{\mathsf{H}} : \mathbb{C} \to \mathbf{Set}$.

4.2 The transpose of *append*

$$append' : \mathsf{List} \mathrel{\dot{\to}} \mathsf{List}^{\mathsf{List}}$$

In Haskell:

$$append' : \forall\, a\,.\, \mathsf{List}\, a \to \forall\, x\,.\, (a \to x) \to (\mathsf{List}\, x \to \mathsf{List}\, x)$$
$$append'\ \ as \ = \ \lambda f \to \lambda bs \to$$
$$\quad append\,(fmap\, f\ as, bs).$$

NB *append′* combines *append* with *fmap*.

4.3 Recall: *nats* and *squares*

$$
\begin{aligned}
&\mathit{nats} : \mathit{Nat} \to \nu\mathsf{Sequ}\\
&\mathit{nats}\ \ n \quad = \mathit{Out}^{\circ}\ (\mathsf{Next}\,(n, \mathit{squares}\,n))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{squares} : \mathit{Nat} \to \nu\mathsf{Sequ}\\
&\mathit{squares}\ \ n \quad = \mathit{Out}^{\circ}\ (\mathsf{Next}\,(n * n, \mathit{nats}\,(n+1)))
\end{aligned}
$$

4.3 Speaking categorically

$$numbers : \langle Nat, Nat \rangle \to \Delta(\nu \mathsf{Sequ})$$

4.3 Adjoints of the diagonal functor

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

$$\mathbb{C} \underset{\Delta}{\overset{+}{\underset{\perp}{\rightleftarrows}}} \mathbb{C} \times \mathbb{C} \underset{\times}{\overset{\Delta}{\underset{\perp}{\rightleftarrows}}} \mathbb{C}$$

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

4.3 Specialising adjoint equations

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

$$\iff$$

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

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

4.3 The transpose of *nats* and *squares*

$$
\begin{aligned}
&\mathit{numbers} : \mathit{Either}\ \mathit{Nat}\ \mathit{Nat} \to \nu\mathsf{Sequ}\\
&\mathit{numbers}\ \ (\mathit{Left}\ \ n) \qquad =\\
&\quad \mathit{Out}^{\circ}\ (\mathsf{Next}\ (n, \mathit{numbers}\ (\mathit{Right}\ n)))\\
&\mathit{numbers}\ \ (\mathit{Right}\ n) \qquad =\\
&\quad \mathit{Out}^{\circ}\ (\mathsf{Next}\ (n * n, \mathit{numbers}\ (\mathit{Left}\ (n + 1))))
\end{aligned}
$$

4.3 A special case: paramorphisms

$$
\begin{aligned}
&\mathit{fac} : \mu\mathsf{Nat} \to \mathit{Nat} \\
&\mathit{fac}\ (\mathit{In}\,(\mathsf{Z})) = 1 \\
&\mathit{fac}\ (\mathit{In}\,(\mathsf{S}\,n)) = \mathit{In}\,(\mathsf{S}\,(\mathit{id}\,n)) * \mathit{fac}\,n
\end{aligned}
$$

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

4.3 A special case: histomorphisms

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

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

4.4 Recall: *sum*

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

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

4.4 Likewise for perfect trees

$$\begin{aligned} &sump : \mathsf{Perfect}\,Nat \to Nat \\ &sump\ (Zero\,n) = n \\ &sump\ (Succ\,p) = sump\,(fmap\,plus\,p) \end{aligned}$$

$$plus\,(a, b) = a + b$$

NB The recursive call is *not* applied to a subterm of *Succ p*.

4.4 Speaking categorically

$$sum : \mathsf{App}_{Nat}\,\mathsf{List} \mathrel{\dot{\to}} \mathsf{K}\,Nat$$

where

$$\begin{aligned}
&\mathsf{App}_X : \mathbb{C}^{\mathbb{D}} \to \mathbb{C} \\
&\mathsf{App}_X\,\mathsf{F} = \mathsf{F}\,X \\
&\mathsf{App}_X\,\alpha = \alpha\,X.
\end{aligned}$$

$$\phi : \forall A\, \mathsf{B} \,.\, \mathbb{C}^{\mathbb{D}}(\mathsf{Lsh}_X\, A, \mathsf{B}) \cong \mathbb{C}(A, \mathsf{App}_X\, \mathsf{B})$$

$$\mathbb{C}^{\mathbb{D}} \underset{\mathsf{App}_X}{\overset{\mathsf{Lsh}_X}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathbb{C} \underset{\mathsf{Rsh}_X}{\overset{\mathsf{App}_X}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathbb{C}^{\mathbb{D}}$$

$$\phi : \forall \mathsf{A}\, B \,.\, \mathbb{C}(\mathsf{App}_X\, \mathsf{A}, B) \cong \mathbb{C}^{\mathbb{D}}(\mathsf{A}, \mathsf{Rsh}_X\, B)$$

4.4 Deriving the left adjoint

$$
\begin{aligned}
& \mathbb{C}(A, \mathsf{App}_X\, \mathsf{B}) \\
\cong\ & \quad \{ \text{ definition of } \mathsf{App}_X \} \\
& \mathbb{C}(A, \mathsf{B}\, X) \\
\cong\ & \quad \{ \text{ Yoneda } \} \\
& \forall\, Y : \mathbb{D} \,.\, \mathbb{D}(X, Y) \to \mathbb{C}(A, \mathsf{B}\, Y) \\
\cong\ & \quad \{ \text{ definition of a copower: } \mathsf{Ix} \to \mathbb{C}(X, Y) \cong \mathbb{C}(\textstyle\sum \mathsf{Ix} \,.\, X, Y) \} \\
& \forall\, Y : \mathbb{D} \,.\, \mathbb{C}(\textstyle\sum \mathbb{D}(X, Y) \,.\, A, \mathsf{B}\, Y) \\
\cong\ & \quad \{ \text{ define } \mathsf{Lsh}_X\, A = \Lambda\, Y : \mathbb{D} \,.\, \textstyle\sum \mathbb{D}(X, Y) \,.\, A \} \\
& \forall\, Y : \mathbb{D} \,.\, \mathbb{C}(\mathsf{Lsh}_X\, A\, Y, \mathsf{B}\, Y) \\
\cong\ & \quad \{ \text{ natural transformation } \} \\
& \mathsf{Lsh}_X\, A \mathrel{\dot{\to}} \mathsf{B}
\end{aligned}
$$

4.4 Left shifts in Haskell

$$\textbf{newtype}\ \mathsf{Lsh}_x\ a\ y = \mathit{Lsh}\ (x \to y, a)$$

$$\textbf{instance}\ \mathit{Functor}\ (\mathsf{Lsh}_x\ a)\ \textbf{where}$$
$$\quad \mathit{fmap}\ f\ (\mathit{Lsh}\ (\kappa, a)) = \mathit{Lsh}\ (f \cdot \kappa, a)$$

$$\phi_{\mathsf{Lsh}} : (\forall\ y\ .\ \mathsf{Lsh}_x\ a\ y \to b\ y) \to (a \to b\ x)$$
$$\phi_{\mathsf{Lsh}}\ \alpha = \lambda s \to \alpha\ (\mathit{Lsh}\ (\mathit{id}, s))$$

$$\phi^{\circ}_{\mathsf{Lsh}} : (\mathit{Functor}\ b) \Rightarrow (a \to b\ x) \to (\forall\ y\ .\ \mathsf{Lsh}_x\ a\ y \to b\ y)$$
$$\phi^{\circ}_{\mathsf{Lsh}}\ g = \lambda(\mathit{Lsh}\ (\kappa, s)) \to \mathit{fmap}\ \kappa\ (g\ s)$$

4.4 Right shifts in Haskell

$$\mathbf{newtype}\ \mathsf{Rsh}_x\, b\, y = \mathit{Rsh}\,\{\mathit{rsh}^{\circ} : (y \to x) \to b\}$$

$$\mathbf{instance}\ \mathit{Functor}\ (\mathsf{Rsh}_x\, b)\ \mathbf{where}$$
$$\quad \mathit{fmap}\, f\, (\mathit{Rsh}\, g) \;=\; \mathit{Rsh}\,(\lambda\kappa \to g\,(\kappa \cdot f))$$

$$\phi_{\mathsf{Rsh}} : (\mathit{Functor}\, a) \Rightarrow (a\, x \to b) \to (\forall\, y\, .\, a\, y \to \mathsf{Rsh}_x\, b\, y)$$
$$\phi_{\mathsf{Rsh}}\, f = \lambda s \to \mathit{Rsh}\,(\lambda\kappa \to f\,(\mathit{fmap}\,\kappa\, s))$$

$$\phi^{\circ}_{\mathsf{Rsh}} : (\forall\, y\, .\, a\, y \to \mathsf{Rsh}_x\, b\, y) \to (a\, x \to b)$$
$$\phi^{\circ}_{\mathsf{Rsh}}\, \beta = \lambda s \to \mathit{rsh}^{\circ}\,(\beta\, s)\, \mathit{id}$$

4.4 Specialising adjoint equations

$$
\begin{aligned}
& x \cdot \mathsf{App}_X\, in = \Psi\, x \\
\iff & \quad \{ \text{ definition of } \mathsf{App}_X \text{ } \} \\
& x \cdot in X = \Psi\, x
\end{aligned}
$$

4.4 The transpose of *sump*

$$
\begin{aligned}
&sump' : \forall\, x \,.\, \mathsf{Perfect}\, x \to (x \to Nat) \to Nat \\
&sump' \quad (Zero\, n) = \lambda \kappa \to \kappa\, n \\
&sump' \quad (Succ\, p) = \lambda \kappa \to sump'\, p\, (plus \cdot (\kappa \times \kappa))
\end{aligned}
$$

4.4 Relation to Generic Haskell

$$
\begin{aligned}
&\mathit{sump}' : \forall x \,.\, (x \to \mathit{Nat}) \to (\mathsf{Perfect}\, x \to \mathit{Nat})\\
&\mathit{sump}' \quad \mathit{sumx} \quad (\mathit{Zero}\, n) \;=\; \mathit{sumx}\, n\\
&\mathit{sump}' \quad \mathit{sumx} \quad (\mathit{Succ}\, p) \;=\\
&\quad \mathit{sump}' \,(\mathit{plus} \cdot (\mathit{sumx} \times \mathit{sumx}))\, p
\end{aligned}
$$

4.5 Recall: *concat*

$$
\begin{aligned}
&\mathit{concat} : \forall\, a \,.\, \mathsf{List}\,(\mathsf{List}\,a) && \to \mathsf{List}\,a \\
&\mathit{concat} \quad (\mathit{Nil}) && = \mathit{Nil} \\
&\mathit{concat} \quad (\mathit{Cons}\,(l, ls)) && = \mathit{append}\,(l, \mathit{concat}\ ls)
\end{aligned}
$$

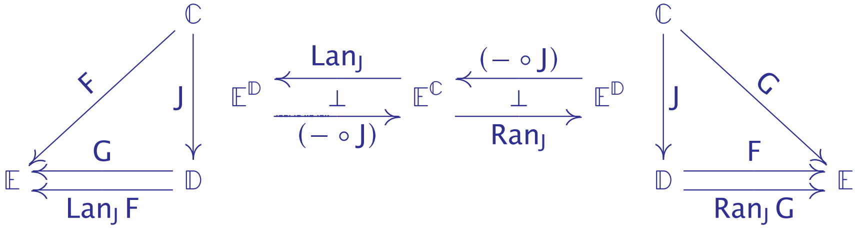
4.5 Speaking categorically

$$concat : \mathsf{Pre}_{\mathsf{List}}\,(\mu\mathsf{LIST}) \mathrel{\dot{\to}} \mathsf{List}$$

where

$$\begin{aligned} \mathsf{Pre}_{\mathsf{J}} &: \mathbb{E}^{\mathbb{D}} \to \mathbb{E}^{\mathbb{C}} \\ \mathsf{Pre}_{\mathsf{J}}\,\mathsf{F} &= \mathsf{F} \circ \mathsf{J} \\ \mathsf{Pre}_{\mathsf{J}}\,\alpha &= \alpha \circ \mathsf{J}. \end{aligned}$$

$$\phi : \forall\, F\, G\, .\; \mathbb{E}^{\mathbb{D}}(\mathsf{Lan}_J\, F, G) \cong \mathbb{E}^{\mathbb{C}}(F, G \circ J)$$



$$\phi : \forall\, F\, G\, .\; \mathbb{E}^{\mathbb{C}}(F \circ J, G) \cong \mathbb{E}^{\mathbb{D}}(F, \mathsf{Ran}_J\, G)$$

$$
\begin{aligned}
& \mathsf{F} \circ \mathsf{J} \mathrel{\dot{\to}} \mathsf{G} \\
\cong\ & \quad \{ \text{natural transformation as an end} \} \\
& \forall A : \mathbb{C} \mathrel{.} \mathbb{E}(\mathsf{F}\,(\mathsf{J}\,A), \mathsf{G}\,A) \\
\cong\ & \quad \{ \text{Yoneda} \} \\
& \forall A : \mathbb{C} \mathrel{.} \forall X : \mathbb{D} \mathrel{.} \mathbb{D}(X, \mathsf{J}\,A) \to \mathbb{E}(\mathsf{F}\,X, \mathsf{G}\,A) \\
\cong\ & \quad \{ \text{definition of power: } \mathsf{Ix} \to \mathbb{C}(A, B) \cong \mathbb{C}(A, \textstyle\prod \mathsf{Ix} \mathrel{.} B) \} \\
& \forall A : \mathbb{C} \mathrel{.} \forall X : \mathbb{D} \mathrel{.} \mathbb{E}(\mathsf{F}\,X, \textstyle\prod \mathbb{D}(X, \mathsf{J}\,A) \mathrel{.} \mathsf{G}\,A) \\
\cong\ & \quad \{ \text{interchange of quantifiers} \} \\
& \forall X : \mathbb{D} \mathrel{.} \forall A : \mathbb{C} \mathrel{.} \mathbb{E}(\mathsf{F}\,X, \textstyle\prod \mathbb{D}(X, \mathsf{J}\,A) \mathrel{.} \mathsf{G}\,A) \\
\cong\ & \quad \{ \text{the functor } \mathbb{E}(\mathsf{F}\,X, -) \text{ preserves ends} \} \\
& \forall X : \mathbb{D} \mathrel{.} \mathbb{E}(\mathsf{F}\,X, \forall A : \mathbb{C} \mathrel{.} \textstyle\prod \mathbb{D}(X, \mathsf{J}\,A) \mathrel{.} \mathsf{G}\,A) \\
\cong\ & \quad \{ \text{define } \mathsf{Ran}_{\mathsf{J}}\,\mathsf{G} = \Lambda X : \mathbb{D} \mathrel{.} \forall A : \mathbb{C} \mathrel{.} \textstyle\prod \mathbb{D}(X, \mathsf{J}\,A) \mathrel{.} \mathsf{G}\,A \} \\
& \forall X : \mathbb{D} \mathrel{.} \mathbb{E}(\mathsf{F}\,X, \mathsf{Ran}_{\mathsf{J}}\,\mathsf{G}\,X) \\
\cong\ & \quad \{ \text{natural transformation as an end} \} \\
& \mathsf{F} \mathrel{\dot{\to}} \mathsf{Ran}_{\mathsf{J}}\,\mathsf{G}
\end{aligned}
$$

4.5 Right Kan extensions in Haskell

$$\mathbf{newtype}\ \mathsf{Ran}_i\, g\, x = \mathit{Ran}\,\{\mathit{ran}^\circ : \forall a\,.\,(x \to i\, a) \to g\, a\}$$

$$\mathbf{instance}\ \mathit{Functor}\ (\mathsf{Ran}_i\, g)\ \mathbf{where}$$
$$\quad \mathit{fmap}\, f\, (\mathit{Ran}\, h) = \mathit{Ran}\,(\lambda\kappa \to h\,(\kappa \cdot f))$$

$$\phi_{\mathsf{Ran}} : (\mathit{Functor}\, f) \Rightarrow (\forall x\,.\, f\,(i\, x) \to g\, x) \to (\forall x\,.\, f\, x \to \mathsf{Ran}_i\, g\, x)$$
$$\phi_{\mathsf{Ran}}\, \alpha = \lambda s \to \mathit{Ran}\,(\lambda\kappa \to \alpha\,(\mathit{fmap}\,\kappa\, s))$$

$$\phi^\circ_{\mathsf{Ran}} : (\forall x\,.\, f\, x \to \mathsf{Ran}_i\, g\, x) \to (\forall x\,.\, f\,(i\, x) \to g\, x)$$
$$\phi^\circ_{\mathsf{Ran}}\, \beta = \lambda s \to \mathit{ran}^\circ\,(\beta\, s)\, \mathit{id}$$

4.5 Left Kan extensions in Haskell

$$\textbf{data}\ \mathsf{Lan}_i\, f\, x = \forall\, a\, .\, \mathit{Lan}\,(i\, a \to x, f\, a)$$

$$\textbf{instance}\ \mathit{Functor}\,(\mathsf{Lan}_i\, f)\ \textbf{where}$$
$$\quad \mathit{fmap}\, f\,(\mathit{Lan}\,(\kappa, s)) = \mathit{Lan}\,(f \cdot \kappa, s)$$

$$\phi_{\mathsf{Lan}} : (\forall\, x\, .\, \mathsf{Lan}_i\, f\, x \to g\, x) \to (\forall\, x\, .\, f\, x \to g\,(i\, x))$$
$$\phi_{\mathsf{Lan}}\, \alpha = \lambda s \to \alpha\,(\mathit{Lan}\,(\mathit{id}, s))$$

$$\phi^{\circ}_{\mathsf{Lan}} : (\mathit{Functor}\, g) \Rightarrow (\forall\, x\, .\, f\, x \to g\,(i\, x)) \to (\forall\, x\, .\, \mathsf{Lan}_i\, f\, x \to g\, x)$$
$$\phi^{\circ}_{\mathsf{Lan}}\, \beta = \lambda(\mathit{Lan}\,(\kappa, s)) \to \mathit{fmap}\, \kappa\,(\beta\, s)$$

4.5 The transpose of *concat*

$$\begin{aligned}&\mathit{concat}' : \forall\, a\, b\,.\; \mu\mathsf{LIST}\, a \to (a \to \mathsf{List}\, b) \to \mathsf{List}\, b\\&\mathit{concat}'\;\; \mathit{as} = \lambda\kappa \to \mathit{concat}\,(\mathit{fmap}\,\kappa\,\mathit{as})\end{aligned}$$

The transpose of *concat* is the bind of the list monad (written $\gg\!=$ in Haskell)!

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	—

Part 5

Application: Type fusion

5.0 Outline

5.1 Memoisation

Say, you want to memoise the function

$$f : Nat \to V$$

so that it caches previously computed values.

Given the interface

$$\mathbf{data}\ \mathsf{Table}\ v$$

$$\begin{aligned} &\mathit{lookup} \quad : \forall\, v\, .\ \mathsf{Table}\ v \to (\mathit{Nat} \to v) \\ &\mathit{tabulate} : \forall\, v\, .\ (\mathit{Nat} \to v) \to \mathsf{Table}\ v, \end{aligned}$$

we can memoize f as follows

$$\begin{aligned} &\mathit{memo\text{-}f} : \mathit{Nat} \to V \\ &\mathit{memo\text{-}f} = \mathit{lookup}\ (\mathit{tabulate}\ f). \end{aligned}$$

5.1 Implementing Table

$$\textbf{data}\ \mathit{Nat} = \mathit{Zero} \mid \mathit{Succ}\ \mathit{Nat}$$

$$\textbf{data}\ \mathsf{Table}\ v = \mathit{Node}\ \{\mathit{zero} : v, \mathit{succ} : \mathsf{Table}\ v\}$$

$$\begin{aligned}
&\mathit{lookup}\ (\mathit{Node}\ \{\mathit{zero} = t\})\ \mathit{Zero} &&= t \\
&\mathit{lookup}\ (\mathit{Node}\ \{\mathit{succ} = t\})\ (\mathit{Succ}\ n) &&= \mathit{lookup}\ t\ n
\end{aligned}$$

$$\begin{aligned}
\mathit{tabulate}\ f = \mathit{Node}\ \{&\mathit{zero} = f\ \mathit{Zero}, \\
&\mathit{succ} = \mathit{tabulate}\ (\lambda n \to f\ (\mathit{Succ}\ n))\}
\end{aligned}$$

5.2 Fusion for adjoint folds

Let $\alpha : \forall X \in \mathbb{D} \,.\, \mathbb{C}(\mathsf{L}\, X, B) \to \mathbb{C}'(\mathsf{L}'\, X, B')$, then

$$\alpha \, (\!|\Psi|\!)_{\mathsf{L}} = (\!|\Psi'|\!)_{\mathsf{L}'} \quad \Longleftarrow \quad \alpha \cdot \Psi = \Psi' \cdot \alpha.$$

NB This subsumes the fusion law for folds.

5.2 Proof of fusion

$$
\begin{aligned}
& \alpha \, (\!|\Psi|\!)_{\mathsf{L}} \cdot \mathsf{L}' \, \mathit{in} \\
= \quad & \{ \text{naturality of } \alpha\text{: } \alpha \, x \cdot \mathsf{L}' \, h = \alpha \, (x \cdot \mathsf{L} \, h) \} \\
& \alpha \, ((\!|\Psi|\!)_{\mathsf{L}} \cdot \mathsf{L} \, \mathit{in}) \\
= \quad & \{ \text{computation} \} \\
& \alpha \, (\Psi \, (\!|\Psi|\!)_{\mathsf{L}}) \\
= \quad & \{ \text{assumption } \alpha \cdot \Psi = \Psi' \cdot \alpha \} \\
& \Psi' \, (\alpha \, (\!|\Psi|\!)_{\mathsf{L}})
\end{aligned}
$$

5.3 Type fusion

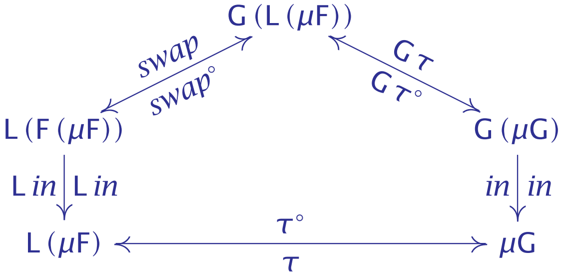
$$\mathbb{C} \underset{\mathsf{G}}{\overset{\mathsf{G}}{\leftrightarrows}} \mathbb{C} \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\perp}{\leftrightarrows}}} \mathbb{D} \underset{\mathsf{F}}{\overset{\mathsf{F}}{\leftrightarrows}} \mathbb{D}$$

$$\mathsf{L}\,(\mu\mathsf{F}) \cong \mu\mathsf{G} \quad \Longleftarrow \quad \mathsf{L} \circ \mathsf{F} \cong \mathsf{G} \circ \mathsf{L}$$

$$\nu\mathsf{F} \cong \mathsf{R}\,(\nu\mathsf{G}) \quad \Longleftarrow \quad \mathsf{F} \circ \mathsf{R} \cong \mathsf{R} \circ \mathsf{G}$$

$$\tau : \mathsf{L}\,(\mu\mathsf{F}) \cong \mu\mathsf{G} \quad \Longleftarrow \quad \mathit{swap} : \mathsf{L} \circ \mathsf{F} \cong \mathsf{G} \circ \mathsf{L}$$

5.3 Definition of τ and τ°



$$\tau \cdot \mathsf{L}\, in = in \cdot \mathsf{G}\, \tau \cdot swap \qquad \text{and} \qquad \tau^\circ \cdot in = \mathsf{L}\, in \cdot swap^\circ \cdot \mathsf{G}\, \tau^\circ$$

5.3 Proof of $\tau \cdot \tau^{\circ} = id_{\mu \mathsf{G}}$

$$
\begin{aligned}
& (\tau \cdot \tau^{\circ}) \cdot in \\
= \quad & \{ \text{definition of } \tau^{\circ} \} \\
& \tau \cdot \mathsf{L}\, in \cdot swap^{\circ} \cdot \mathsf{G}\, \tau^{\circ} \\
= \quad & \{ \text{definition of } \tau \} \\
& in \cdot \mathsf{G}\, \tau \cdot swap \cdot swap^{\circ} \cdot \mathsf{G}\, \tau^{\circ} \\
= \quad & \{ \text{inverses} \} \\
& in \cdot \mathsf{G}\, \tau \cdot \mathsf{G}\, \tau^{\circ} \\
= \quad & \{ \mathsf{G} \text{ functor} \} \\
& in \cdot \mathsf{G}\, (\tau \cdot \tau^{\circ})
\end{aligned}
$$

The equation $x \cdot in = in \cdot \mathsf{G}\, x$ has a unique solution. Since id is also a solution, the result follows.

5.3 Proof of $\tau^\circ \cdot \tau = id_{\mathsf{L}\,(\mu\mathsf{F})}$

$$
\begin{aligned}
& (\tau^\circ \cdot \tau) \cdot \mathsf{L}\, in \\
= {} & \quad \{ \text{definition of } \tau \} \\
& \tau^\circ \cdot in \cdot \mathsf{G}\,\tau \cdot swap \\
= {} & \quad \{ \text{definition of } \tau^\circ \} \\
& \mathsf{L}\, in \cdot swap^\circ \cdot \mathsf{G}\,\tau^\circ \cdot \mathsf{G}\,\tau \cdot swap \\
= {} & \quad \{ \mathsf{G} \text{ functor} \} \\
& \mathsf{L}\, in \cdot swap^\circ \cdot \mathsf{G}\,(\tau^\circ \cdot \tau) \cdot swap
\end{aligned}
$$

Again, $x \cdot \mathsf{L}\, in = \mathsf{L}\, in \cdot swap^\circ \cdot \mathsf{G}\, x \cdot swap$ has a unique solution.
And again, *id* is also solution, which implies the result.

5.4 Application: firstification

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

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

$$\mathsf{List}\ \mathit{Nat} \quad \cong \quad \mathit{Stack}$$

5.4 Speaking categorically

$$\mathsf{App}_{Nat}\,(\mu\mathsf{LIST}) \cong \mu\mathsf{Stack}$$

$$\Longleftarrow$$

$$\mathsf{App}_{Nat} \circ \mathsf{LIST} \cong \mathsf{Stack} \circ \mathsf{App}_{Nat}$$

5.4 Proof of $\mathsf{App}_{Nat} \circ \mathsf{LIST} \cong \mathsf{Stack} \circ \mathsf{App}_{Nat}$

$$
\begin{aligned}
& \mathsf{App}_{Nat} \circ \mathsf{LIST} \\
\cong \quad & \{ \text{ composition of functors and definition of } \mathsf{App} \text{ } \} \\
& \Lambda X \mathbin{.} \mathsf{LIST}\, X\, Nat \\
\cong \quad & \{ \text{ definition of } \mathsf{LIST} \text{ } \} \\
& \Lambda X \mathbin{.} 1 + Nat \times X\, Nat \\
\cong \quad & \{ \text{ definition of } \mathsf{Stack} \text{ } \} \\
& \Lambda X \mathbin{.} \mathsf{Stack}\, (X\, Nat) \\
\cong \quad & \{ \text{ composition of functors and definition of } \mathsf{App} \text{ } \} \\
& \mathsf{Stack} \circ \mathsf{App}_{Nat}
\end{aligned}
$$

5.4 In Haskell

$$
\begin{aligned}
&\mathit{swap} : \forall\, x\,.\ \ \mathsf{LIST}\; x\; \mathit{Nat} \to \mathsf{Stack}\,(x\; \mathit{Nat})\\
&\mathit{swap}\ \ \mathsf{Nil} = \mathsf{Empty}\\
&\mathit{swap}\ \ (\mathsf{Cons}\,(n, x)) = \mathsf{Push}\,(n, x)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{swap}^{\circ} : \forall\, x\,.\ \mathsf{Stack}\,(x\; \mathit{Nat}) \to \mathsf{LIST}\; x\; \mathit{Nat}\\
&\mathit{swap}^{\circ}\ \ \mathsf{Empty} = \mathsf{Nil}\\
&\mathit{swap}^{\circ}\ \ (\mathsf{Push}\,(n, x)) = \mathsf{Cons}\,(n, x)
\end{aligned}
$$

$$
\begin{aligned}
&\Lambda\text{-}\mathit{lift} : \mu\mathsf{Stack} \to \mu\mathsf{LIST}\; \mathit{Nat}\\
&\Lambda\text{-}\mathit{lift}\ \ (\mathit{In}\; x) = \mathit{In}\,(\mathit{swap}^{\circ}\,(\mathit{fmap}\; \Lambda\text{-}\mathit{lift}\; x))
\end{aligned}
$$

$$
\begin{aligned}
&\Lambda\text{-}\mathit{drop} : \mu\mathsf{LIST}\; \mathit{Nat} \to \mu\mathsf{Stack}\\
&\Lambda\text{-}\mathit{drop}\ \ (\mathit{In}\; x) = \mathit{In}\,(\mathit{fmap}\; \Lambda\text{-}\mathit{drop}\,(\mathit{swap}\; x))
\end{aligned}
$$

5.5 Application: type specialisation

Lists of optional values, $\mathsf{List} \circ \mathsf{Maybe}$ with

$$\mathbf{data}\ \mathsf{Maybe}\ a = \mathit{Nothing} \mid \mathit{Just}\ a,$$

can be represented more compactly using the tailor-made

$$\mathbf{data}\ \mathsf{Sequ}\ a = \mathit{Done} \mid \mathit{Skip}\ (\mathsf{Sequ}\ a) \mid \mathit{Yield}\ (a, \mathsf{Sequ}\ a).$$

5.5 Speaking categorically

$$\mathsf{List} \circ \mathsf{Maybe} \quad \cong \quad \mathsf{Sequ},$$

$$\mathsf{Pre}_{\mathsf{Maybe}}\,(\mu\mathsf{LIST}) \cong \mu\mathsf{SEQU}$$

$$\Longleftarrow$$

$$\mathsf{Pre}_{\mathsf{Maybe}} \circ \mathsf{LIST} \cong \mathsf{SEQU} \circ \mathsf{Pre}_{\mathsf{Maybe}}$$

5.5 Proof of $\mathsf{Pre}_{\mathsf{Maybe}} \circ \mathsf{LIST} \cong \mathsf{SEQU} \circ \mathsf{Pre}_{\mathsf{Maybe}}$

$$
\begin{aligned}
& \mathsf{LIST}\,X \circ \mathsf{Maybe} \\
\cong\quad & \{\ \text{composition of functors}\ \} \\
& \Lambda A\,.\,\mathsf{LIST}\,X\,(\mathsf{Maybe}\,A) \\
\cong\quad & \{\ \text{definition of } \mathsf{LIST}\ \} \\
& \Lambda A\,.\,1 + \mathsf{Maybe}\,A \times X\,(\mathsf{Maybe}\,A) \\
\cong\quad & \{\ \text{definition of } \mathsf{Maybe}\ \} \\
& \Lambda A\,.\,1 + (1 + A) \times X\,(\mathsf{Maybe}\,A) \\
\cong\quad & \{\ \times \text{ distributes over } + \text{ and } 1 \times B \cong B\ \} \\
& \Lambda A\,.\,1 + X\,(\mathsf{Maybe}\,A) + A \times X\,(\mathsf{Maybe}\,A) \\
\cong\quad & \{\ \text{composition of functors}\ \} \\
& \Lambda A\,.\,1 + (X \circ \mathsf{Maybe})\,A + A \times (X \circ \mathsf{Maybe})\,A \\
\cong\quad & \{\ \text{definition of } \mathsf{SEQU}\ \} \\
& \mathsf{SEQU}\,(X \circ \mathsf{Maybe})
\end{aligned}
$$

5.5 In Haskell

$$
\begin{aligned}
&\mathit{swap} : \forall x \,.\, \forall a \,. \\
&\qquad \mathsf{LIST}\; x\; (\mathsf{Maybe}\; a) \to \mathsf{SEQU}\; (x \circ \mathsf{Maybe})\; a \\
&\mathit{swap}\; (\mathsf{Nil}) = \mathsf{Done} \\
&\mathit{swap}\; (\mathsf{Cons}\; (\mathit{Nothing}, x)) = \mathsf{Skip}\; x \\
&\mathit{swap}\; (\mathsf{Cons}\; (\mathit{Just}\; a, x)) = \mathsf{Yield}\; (a, x)
\end{aligned}
$$

5.6 Recall *Nat* and Table

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

$$\mathbf{data}\ \mathsf{Table}\ \mathit{val} = \mathit{Node}\ \{\mathit{zero} : \mathit{val}, \mathit{succ} : \mathsf{Table}\ \mathit{val}\}$$

$$V^{\mathit{Nat}} \cong \mathsf{Table}\ V$$

$$(-)^{Nat} \cong \mathsf{Table}$$

5.6 Truth tables

$$(\wedge) : \mathit{Bool}^{\mathit{Bool} \times \mathit{Bool}}$$

False	*False*
False	*True*

$$(-)^{Bool \times Bool} \quad \cong \quad (\mathsf{Id} \mathbin{\dot{\times}} \mathsf{Id}) \mathbin{\dot{\times}} (\mathsf{Id} \mathbin{\dot{\times}} \mathsf{Id})$$

5.6 Laws of exponentials

$$
\begin{aligned}
V^{0} &\cong 1 \\
V^{1} &\cong V \\
V^{A+B} &\cong V^{A} \times V^{B} \\
V^{A\times B} &\cong (V^{B})^{A}
\end{aligned}
$$

5.6 Curried exponentiation

$$\mathsf{Exp} : \mathbb{C} \to (\mathbb{C}^{\mathbb{C}})^{\mathsf{op}}$$

$$\mathsf{Exp}\,\mathsf{K} = \Lambda V \,.\, V^{\mathsf{K}}$$

$$\mathsf{Exp}\,f = \Lambda V \,.\, (id_V)^{f}$$

5.6 Laws of exponentials

$$
\begin{aligned}
\mathsf{Exp}\,0 &\cong \mathsf{K}\,1 \\
\mathsf{Exp}\,1 &\cong \mathsf{Id} \\
\mathsf{Exp}\,(A + B) &\cong \mathsf{Exp}\,A \mathbin{\dot{\times}} \mathsf{Exp}\,B \\
\mathsf{Exp}\,(A \times B) &\cong \mathsf{Exp}\,A \cdot \mathsf{Exp}\,B
\end{aligned}
$$

5.6 Exp is a left adjoint

$$(\mathbb{C}^{\mathbb{C}})^{\mathsf{op}} \underset{\mathsf{G}}{\overset{\mathsf{G}}{\leftrightarrows}} (\mathbb{C}^{\mathbb{C}})^{\mathsf{op}} \underset{\mathsf{Sel}}{\overset{\mathsf{Exp}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathbb{C} \underset{\mathsf{F}}{\overset{\mathsf{F}}{\leftrightarrows}} \mathbb{C}$$

5.6 Deriving the right adjoint

$$
\begin{aligned}
& (\mathbb{C}^{\mathbb{C}})^{\mathsf{op}}(\mathsf{Exp}\,A, B) \\
\cong\quad & \{\ \text{definition of } -^{\mathsf{op}}\ \} \\
& \mathbb{C}^{\mathbb{C}}(B, \mathsf{Exp}\,A) \\
\cong\quad & \{\ \text{natural transformation as an end}\ \} \\
& \forall\, X \in \mathbb{C} \,.\, \mathbb{C}(B\,X, \mathsf{Exp}\,A\,X) \\
\cong\quad & \{\ \text{definition of } \mathsf{Exp}\ \} \\
& \forall\, X \in \mathbb{C} \,.\, \mathbb{C}(B\,X, X^{A}) \\
\cong\quad & \{\ - \times Y \dashv (-)^{Y} \text{ and } Y \times Z \cong Z \times Y\ \} \\
& \forall\, X \in \mathbb{C} \,.\, \mathbb{C}(A, X^{B\,X}) \\
\cong\quad & \{\ \text{the functor } \mathbb{C}(A, -) \text{ preserves ends}\ \} \\
& \mathbb{C}(A, \forall\, X \in \mathbb{C} \,.\, X^{B\,X}) \\
\cong\quad & \{\ \text{define } \mathsf{Sel}\,B = \forall\, X \in \mathbb{C} \,.\, X^{B\,X}\ \} \\
& \mathbb{C}(A, \mathsf{Sel}\,B)
\end{aligned}
$$

$$(\mathbb{C}^{\mathbb{C}})^{\mathsf{op}} \underset{\mathsf{G}}{\overset{\mathsf{G}}{\leftrightarrows}} (\mathbb{C}^{\mathbb{C}})^{\mathsf{op}} \underset{\mathsf{Sel}}{\overset{\mathsf{Exp}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathbb{C} \underset{\mathsf{F}}{\overset{\mathsf{F}}{\leftrightarrows}} \mathbb{C}$$

Since Exp is a contra-variant functor, τ and *swap* live in an opposite category. Type fusion in terms of arrows in $\mathbb{C}^{\mathbb{C}}$:

$$\tau : \nu\mathsf{G} \cong \mathsf{Exp}\,(\mu\mathsf{F}) \quad \Longleftarrow \quad \mathit{swap} : \mathsf{G} \circ \mathsf{Exp} \cong \mathsf{Exp} \circ \mathsf{F}.$$

5.6 Look-up and tabulate

The isomorphism $\tau : \nu \mathsf{G} \dot{\rightarrow} \mathsf{Exp}\,(\mu \mathsf{F})$ is a curried *look-up* function that maps a memo table to an exponential.

$$lookup\,(Out^{\circ}\ t)\ (in\, i) = swap\,(\mathsf{G}\, lookup\, t)\ i$$

The inverse $\tau^{\circ} : \mathsf{Exp}\,(\mu \mathsf{F}) \dot{\rightarrow} \nu \mathsf{G}$ is a transformation that *tabulates* a given exponential.

$$tabulate\, f = Out^{\circ}\ (\mathsf{G}\, tabulate\,(swap^{\circ}\ (f \cdot in)))$$

5.6 In Haskell

The transformation *swap* implements $V \times V^X \cong V^{1+X}$.

$$\begin{aligned}
&\mathit{swap} : \forall\, x \,.\, \forall\, \mathit{val} \,.\, \mathsf{TABLE}\,(\mathsf{Exp}\, x)\, \mathit{val} \to (\mathsf{Nat}\, x \to \mathit{val})\\
&\mathit{swap}\,(\mathsf{Node}\,(v, t))\,(\mathsf{Zero}) \quad = v\\
&\mathit{swap}\,(\mathsf{Node}\,(v, t))\,(\mathsf{Succ}\, n) = t\, n
\end{aligned}$$

The inverse of *swap* implements $V^{1+X} \cong V \times V^X$.

$$\begin{aligned}
&\mathit{swap}^{\circ} : \forall\, x \,.\, \forall\, \mathit{val} \,.\, (\mathsf{Nat}\, x \to \mathit{val}) \to \mathsf{TABLE}\,(\mathsf{Exp}\, x)\, \mathit{val}\\
&\mathit{swap}^{\circ}\, \mathsf{f} = \mathsf{Node}\,(\mathsf{f}\, \mathsf{Zero}, \mathsf{f} \cdot \mathsf{Succ})
\end{aligned}$$

5.6 In Haskell

$$\mathit{lookup} : \forall\, \mathit{val}\,.\, \nu\mathsf{TABLE}\,\mathit{val} \to (\mu\mathsf{Nat} \to \mathit{val})$$
$$\mathit{lookup}\,(\mathit{Out}^{\circ}\,(\mathsf{Node}\,(v, t)))\,(\mathit{In}\,\mathsf{Zero}) = v$$
$$\mathit{lookup}\,(\mathit{Out}^{\circ}\,(\mathsf{Node}\,(v, t)))\,(\mathit{In}\,(\mathsf{Succ}\,n)) = \mathit{lookup}\,t\,n$$

$$\mathit{tabulate} : \forall\, \mathit{val}\,.\, (\mu\mathsf{Nat} \to \mathit{val}) \to \nu\mathsf{TABLE}\,\mathit{val}$$
$$\mathit{tabulate}\,f = \mathit{Out}^{\circ}\,(\mathsf{Node}\,(\mathsf{f}\,(\mathit{In}\,\mathsf{Zero}), \mathit{tabulate}\,(f \cdot \mathit{In} \cdot \mathsf{Succ})))$$

Part 6

Epilogue

6.0 Summary

- Adjoint (un-) folds capture many recursion schemes.
- Adjunctions play a central role.
- Tabulation is an intriguing example.

6.0 Limitations

- Simultaneous recursion doesn't fit under the umbrella.

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

- 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.

Part 7

Tagless interpreters

7.0 Initial algebras: view from the left

$$\mathbf{data}\ Expr_0 = Lit\ Nat \mid Add\ (Expr_0, Expr_0)$$

$$e_0 : Expr_0$$
$$e_0 = Add\ (Lit\ 4700, Lit\ 11)$$

$$\textbf{data}\ \mathsf{Expr}\ expr = \mathsf{Lit}\ Nat \mid \mathsf{Add}\ (expr, expr)$$

The evaluation algebra.

$$\begin{aligned} &eval_0 : \mathsf{Expr}\ Nat \to Nat \\ &eval_0\ (\mathsf{Lit}\ n) = n \\ &eval_0\ (\mathsf{Add}\ (n_1, n_2)) = n_1 + n_2 \end{aligned}$$

The fold for expressions.

$$\begin{aligned} &fold_0 : \forall\ val\ .\ (\mathsf{Expr}\ val \to val) \to (Expr_0 \to val) \\ &fold_0\ alg\ (Lit\ n) = alg\ (\mathsf{Lit}\ n) \\ &fold_0\ alg\ (Add\ (e_1, e_2)) = alg\ (\mathsf{Add}\ (fold_0\ alg\ e_1, fold_0\ alg\ e_2)) \end{aligned}$$

$$
\begin{aligned}
& \mathbb{C}(\mathsf{Expr}\,\mathit{Val}, \mathit{Val}) \\
\cong\quad & \{ \text{ definition of } \mathsf{Expr} \} \\
& \mathbb{C}(\mathit{Nat} + (\mathit{Val} \times \mathit{Val}), \mathit{Val}) \\
\cong\quad & \{ (+) \dashv \Delta \} \\
& (\mathbb{C} \times \mathbb{C})(\langle \mathit{Nat}, \mathit{Val} \times \mathit{Val} \rangle, \Delta \mathit{Val}) \\
\cong\quad & \{ \text{ product categories } \} \\
& \mathbb{C}(\mathit{Nat}, \mathit{Val}) \times \mathbb{C}(\mathit{Val} \times \mathit{Val}, \mathit{Val})
\end{aligned}
$$

$$\textbf{data}\ \mathit{Alg}\ \mathit{val} = \mathit{Alg}\ \{\mathit{lit} : \mathit{Nat} \to \mathit{val}, \mathit{add} : (\mathit{val}, \mathit{val}) \to \mathit{val}\}$$

$$\textbf{data}\ Alg\ val = Alg\,\{\, lit : Nat \to val, add : (val, val) \to val \,\}$$

The evaluation algebra.

$$\begin{aligned} eval_0 &: Alg\ Nat \\ eval_0 &= Alg\,\{\, lit = id, add = \lambda(n_1, n_2) \to n_1 + n_2 \,\} \end{aligned}$$

The fold for expressions.

$$\begin{aligned} &fold_0 : \forall\ val\ .\ Alg\ val \to (Expr_0 \to val) \\ &fold_0\ alg\ (Lit\ n) = lit\ alg\ n \\ &fold_0\ alg\ (Add\ (e_1, e_2)) = add\ alg\ (fold_0\ alg\ e_1, fold_0\ alg\ e_2) \end{aligned}$$

7.0 Initial algebras: view from the right

$$
\begin{aligned}
& \forall\, val \,.\, Alg\, val \to (Expr_0 \to val) \\
\cong\ & \quad \{\, A \to (B \to C) \cong B \to (A \to C) \,\} \\
& \forall\, val \,.\, Expr_0 \to (Alg\, val \to val) \\
\cong\ & \quad \{\, \forall\, a \,.\, A \to \mathsf{T}\, a \cong A \to \forall\, a \,.\, \mathsf{T}\, a \,\} \\
& Expr_0 \to (\forall\, val \,.\, Alg\, val \to val)
\end{aligned}
$$

7.0 . . . using records

$$\textbf{newtype}\ Expr_1 = Expr\ \{ fold_1 : \forall\ val\ .\ Alg\ val \rightarrow val \}$$

$$e_1 : Expr_1$$
$$e_1 = Expr\ \{ fold_1 = \lambda alg \rightarrow add\ alg\ (lit\ alg\ 4700, lit\ alg\ 11) \}$$

Converting between the left and right view.

$$toRight : Expr_0 \rightarrow Expr_1$$
$$toRight\ e = Expr\ \{ fold_1 = \lambda i \rightarrow fold_0\ i\ e \}$$

$$toLeft : Expr_1 \rightarrow Expr_0$$
$$toLeft\ e = fold_1\ e\ (Alg\ \{ lit = Lit, add = Add \})$$

7.0 ...using classes

$$\textbf{class}\ Alg\ val\ \textbf{where}\ \{ lit : Nat \to val; add : (val, val) \to val \}$$

$$e_1 : (Alg\ val) \Rightarrow val$$
$$e_1 = add\ (lit\ 4700, lit\ 11)$$

Converting between the left and right view.

$$toRight : (Alg\ val) \Rightarrow Expr_0 \to val$$
$$toRight\ e = fold_0\ (Alg\ \{ lit = lit, add = add \})\ e$$

$$\textbf{instance}\ Alg\ Expr_0\ \textbf{where}\ \{ lit = Lit; add = Add \}$$

$$toLeft : Expr_0 \to Expr_0$$
$$toLeft\ e = e$$

7.0 Parametricity $\forall A . (\mathsf{F}\, A \to A) \to A$

$$
\begin{aligned}
& (\phi, \phi) \in \forall A . (\mathsf{F}\, A \to A) \to A \\
\iff & \quad \{ \text{ polymorphic type } \} \\
& \forall h . (\phi\, A_1, \phi\, A_2) \in (\mathsf{F}\, h \to h) \to h \\
\iff & \quad \{ \text{ function type } \} \\
& \forall h . \forall f_1\, f_2 . (f_1, f_2) \in \mathsf{F}\, h \to h \Longrightarrow (\phi\, A_1\, f_1, \phi\, A_2\, f_2) \in h \\
\iff & \quad \{ \text{ base } \} \\
& \forall h . \forall f_1\, f_2 . (f_1, f_2) \in \mathsf{F}\, h \to h \Longrightarrow h\,(\phi\, A_1\, f_1) = \phi\, A_2\, f_2 \\
\iff & \quad \{ \text{ function type } \} \\
& \forall h . \forall f_1\, f_2 . h \cdot f_1 = f_2 \cdot \mathsf{F}\, h \Longrightarrow h\,(\phi\, A_1\, f_1) = \phi\, A_2\, f_2
\end{aligned}
$$

7.0 An isomorphism

$$
\begin{aligned}
\mathit{toRight}\ i &= \lambda a\,.\,(\!|a|\!)\ i \\
\mathit{toLeft}\ f &= f\ \mathit{in}
\end{aligned}
$$

$$
\begin{aligned}
& \mathit{toLeft}\ (\mathit{toRight}\ i) \\
={}& \quad \{\ \text{definition of } \mathit{toRight}\ \} \\
& \mathit{toLeft}\ (\lambda a\,.\,(\!|a|\!)\ i) \\
={}& \quad \{\ \text{definition of } \mathit{toLeft}\ \} \\
& (\!|\mathit{in}|\!)\ i \\
={}& \quad \{\ \text{reflection}\ \} \\
& i
\end{aligned}
$$

$$
\begin{aligned}
\mathit{toRight}\ i &= \lambda a\,.\,(\!|a|\!)\ i \\
\mathit{toLeft}\ f &= f\ \mathit{in}
\end{aligned}
$$

$$
\begin{aligned}
& \mathit{toRight}\ (\mathit{toLeft}\ f) \\
=\ & \quad \{\ \text{definition of } \mathit{toLeft}\ \} \\
& \mathit{toRight}\ (f\ \mathit{in}) \\
=\ & \quad \{\ \text{definition of } \mathit{toRight}\ \} \\
& \lambda a\,.\,(\!|a|\!)\ (f\ \mathit{in}) \\
=\ & \quad \{\ \text{parametricity with } (\!|a|\!) \cdot \mathit{in} = a \cdot \mathsf{F}\,(\!|a|\!)\ \} \\
& \lambda a\,.\,f\ a \\
=\ & \quad \{\ \text{extensionality}\ \} \\
& f
\end{aligned}
$$