Unifying Structured Recursion Schemes

Ralf Hinze Nicolas Wu Jeremy Gibbons

Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, England
{ralf.hinze,nicolas.wu,jeremy.gibbons}@cs.ox.ac.uk

Abstract

Folds over inductive datatypes are well understood and widely used. In their plain form, they are quite restricted; but many disparate generalisations have been proposed that enjoy similar calculational benefits. There have also been attempts to unify the various generalisations: two prominent such unifications are the 'recursion schemes from comonads' of Uustalu, Vene and Pardo, and our own 'adjoint folds'. Until now, these two unified schemes have appeared incompatible. We show that this appearance is illusory: in fact, adjoint folds subsume recursion schemes from comonads. The proof of this claim involves standard constructions in category theory that are nevertheless not well known in functional programming: Eilenberg-Moore categories and bialgebras.

Categories and Subject Descriptors D.1.1 [*Programming Techniques*]: Applicative (Functional) Programming; D.3.2 [*Programming Languages*]: Language Classifications—applicative (functional) languages; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—algebraic approaches to semantics

General Terms Languages; Theory; Verification

Keywords recursion schemes; adjunctions; comonads; bialgebras; distributive laws

1. Introduction

It has long been understood that explicit recursion is the 'goto' of pure functional programming [22], and should be considered harmful to program comprehension and analysis. Instead, structured recursion operators such as catamorphisms (folds) [10, 19] ought to be used wherever possible: they make termination manifest, and enjoy many useful calculational properties which would otherwise have to be established afresh for each new application.

However, catamorphisms are relatively restricted. There are many other structured patterns of recursion, equally harmless and worth capturing, that do not quite fit the scheme. Variations that have been proposed in the past include folds with parameters and accumulating folds [25], which may depend on constant or varying additional arguments; mutumorphisms [8], which are pairs of mutually recursive functions; zygomorphisms [18], which consist of a main recursive function and an auxiliary one on which it depends; paramorphisms [21], in which the body of structural recursion has access to immediate subterms as well as to their images under the recursion; histomorphisms [26], in which the body has access to the recursive images of all subterms, not just the immediate ones; and so-called generalised folds [4], which use polymorphic recursion to handle nested datatypes.

The many divergent generalisations of catamorphisms can be bewildering to the uninitiated, and there have been attempts to unify them. One approach is the identification of recursion schemes from comonads [30] (which we call 'rsfcs' for short). Comonads capture the general idea of 'evaluation in context' [27], and rsfcs make contextual information available to the body of the recursion. This pattern subsumes zygomorphisms and histomorphisms.

A more recent attempt [11] uses adjunctions as the common thread. Adjoint folds arise by inserting a left adjoint functor into the recursive characterisation, thereby adapting the form of the recursion; they subsume accumulating folds, mutumorphisms (and hence zygomorphisms), and generalised folds.

Given that adjoint folds and rsfcs cover some of the same examples, it seems reasonable to suspect a deeper relationship between them. That suspicion is strengthened by the observation that every adjunction induces a comonad, and every comonad can be factored into adjoint functors. And indeed, the suspicion turns out to be well founded. In this paper, we show that rsfcs are subsumed by adjoint folds. Moreover, although the converse does not hold, we identify those adjoint folds that correspond to rsfcs.

Technically speaking, our contributions are as follows:

- We provide a fresh account of adjoint folds, making essential use of liftings and conjugates. Very briefly, adjoint folds are parametrised by an adjunction $\mathsf{L} \dashv \mathsf{R}$ and a distributive law $\sigma : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L}$ that connects data structure to control structure.
- We show that that rsfcs [30] are subsumed by adjoint folds.
- We state precisely the relationship to the (type) fusion rule of categorical fixed-point calculus [1]. In essence, type fusion allows us to fuse an application of a left adjoint with an initial algebra to form another initial algebra, $\mathsf{L}\,(\mu\mathsf{C}) \cong \mu\mathsf{D}$, under the stronger assumption that σ is an isomorphism.
- We prove that adjoint folds can be framed as rsfcs, if a distributive isomorphism σ exists.

We dissect most of the proofs into two parts: first, we establish a bijection between certain arrows and homomorphisms; second, we instantiate the bijections to initial or free algebras.

The unified approach to recursion schemes is based on adjoint folds and unfolds, so no new theory is needed. This is good news indeed! The message of this paper is that the existing theory is more general than we anticipated. The unification is more than merely an intellectual curiosity: it promises concrete returns, too—for example, through general techniques for combining different recursion schemes (most functions actually use a combination of recursion schemes).

ICFP '13, September 25–27, 2013, Boston, MA, USA.

ACM 978-1-4503-2326-0/13/09…$15.00.
http://dx.doi.org/10.1145/2500365.2500578

Because of space limitations, we focus on algebras and inductive types; but everything dualises elegantly to a theory of adjoint unfolds [11] that subsumes patterns such as apomorphisms [31] and futumorphisms [26]; we return to this point in Section 9.

The paper is organised as follows: Section 2 presents a smörgåsbord of Haskell examples, which are picked up later; Section 3 summarises some of the theoretical background; Section 4 walks through a basic example of our unifying theory of adjoint folds, which is set out in Section 5; Section 6 shows that all rsfcs are adjoint folds, and Section 7 identifies those adjoint folds that are rsfcs; and finally, Section 8 discusses related work, and Section 9 concludes and points out directions for future work.

2. A Zoo of Morphisms

In this section we exhibit a number of specimens from the zoo of morphisms, which will serve to illustrate the theory that follows. We use Haskell as a lingua franca for codifying our categorical constructions as programs. However, throughout the paper we are careful to distinguish between inductive and coinductive types, which Haskell conflates.

Catamorphism The most basic recursion scheme is the *catamorphism*, known more colloquially as *fold*. A catamorphism decomposes an inductively defined structure, replacing each of the constructors with a provided argument. An example of this pattern is to compute the depth of a binary tree.

$$\begin{aligned}&\textbf{data}\ \mathit{Tree} = \mathit{Empty} \mid \mathit{Node}\ \mathit{Tree}\ \mathbb{N}\ \mathit{Tree}\\ &\mathit{depth} :: \mathit{Tree} \to \mathbb{N}\\ &\mathit{depth}\ (\mathit{Empty}) = 0\\ &\mathit{depth}\ (\mathit{Node}\ l\ a\ r) = 1 + (\mathit{depth}\ l\ `\mathit{max}`\ \mathit{depth}\ r)\end{aligned}$$

Folds with parameters Folds with constant parameters take an additional argument, on which results may depend. List concatenation is a canonical example:

$$\begin{aligned}&\mathit{cat} :: ([a],[a]) \to [a]\\ &\mathit{cat}\ ([\,], ys) = ys\\ &\mathit{cat}\ (x : xs, ys) = x : \mathit{cat}\ (xs, ys)\ .\end{aligned}$$

Here, the second component of the pair is the parameter; *cat* is not just a fold because the pair argument is not of an an inductive type.

In folds with accumulating parameters, the additional argument may vary in recursive calls. Haskell's *foldl* is an example. More interesting examples are provided by downwards accumulations on trees [9]; for example, replacing every element with its depth (if the accumulator is initialised to 0):

$$\begin{aligned}&\mathit{depths} :: (\mathit{Tree}, \mathbb{N}) \to \mathit{Tree}\\ &\mathit{depths}\ (\mathit{Empty}, n) = \mathit{Empty}\\ &\mathit{depths}\ (\mathit{Node}\ l\ a\ r, n) = \mathit{Node}\ (\mathit{depths}\ (l, n{+}1))\ n\ (\mathit{depths}\ (r, n{+}1))\ .\end{aligned}$$

This is a rather simple example; in general, the accumulating parameter will vary in different ways in different branches.

Paramorphism The *paramorphism* models primitive recursion: the body has access not only to the results of recursive calls, but also to the substructures on which these calls are made. An example of a paramorphism is counting the words in a string:

$$\begin{aligned}&\mathit{wc} :: [\mathit{Char}] \to \mathit{Int}\\ &\mathit{wc}\ [\,] = 0\\ &\mathit{wc}\ (c : cs)\\ &\quad \mid \neg\ (\mathit{isSpace}\ c) \wedge (\mathit{null}\ cs \vee \mathit{isSpace}\ (\mathit{head}\ cs)) = \mathit{wc}\ cs + 1\\ &\quad \mid \mathit{otherwise} = \mathit{wc}\ cs\ .\end{aligned}$$

Note that in the clause for non-empty lists, the result depends not only on a recursive call *wc cs* on the substructure, but also on the substructure *cs* itself.

Zygomorphism A variation is the *zygomorphism*, where the recursion is aided by an auxiliary function:

$$\begin{aligned}&\mathit{perfect} :: \mathit{Tree} \to \mathbb{B}\\ &\mathit{perfect}\ \mathit{Empty} = \mathit{True}\\ &\mathit{perfect}\ (\mathit{Node}\ l\ a\ r) = \mathit{perfect}\ l \wedge \mathit{perfect}\ r \wedge (\mathit{depth}\ l == \mathit{depth}\ r)\ .\end{aligned}$$

The function *perfect* is not a simple fold, since it relies on an auxiliary traversal of the tree structure using *depth*.

Mutumorphism A *mutumorphism* generalises the idea of a zygomorphism, allowing the recursive functions to rely mutually on one another. For example, consider the *odd* and *even* functions:

$$\begin{aligned}&\mathit{odd} :: \mathbb{N} \to \mathbb{B} \qquad \mathit{even} :: \mathbb{N} \to \mathbb{B}\\ &\mathit{odd}\ 0 = \mathit{False} \qquad \mathit{even}\ 0 = \mathit{True}\\ &\mathit{odd}\ (n+1) = \mathit{even}\ n \qquad \mathit{even}\ (n+1) = \mathit{odd}\ n\ .\end{aligned}$$

Here, the functions work as a pair in tandem as they recurse through the structure of natural numbers.

Nested datatypes Functions over nested datatypes such as perfect trees or random-access lists have to deal with the polymorphic recursion. For example, consider summing a perfect tree of numbers:

$$\begin{aligned}&\textbf{data}\ \mathit{Perfect}\ a = \mathit{Zero}\ a \mid \mathit{Succ}\ (\mathit{Perfect}\ (a,a))\\ &\textbf{instance}\ \mathit{Functor}\ \mathit{Perfect}\ \textbf{where}\\ &\quad \mathit{fmap}\ f\ (\mathit{Zero}\ a) = \mathit{Zero}\ (f\ a)\\ &\quad \mathit{fmap}\ f\ (\mathit{Succ}\ p) = \mathit{Succ}\ (\mathit{fmap}\ (\lambda(x,y) \to (f\,x, f\,y))\ p)\\ &\mathit{total} :: \mathit{Perfect}\ \mathbb{N} \to \mathbb{N}\\ &\mathit{total}\ (\mathit{Zero}\ n) = n\\ &\mathit{total}\ (\mathit{Succ}\ p) = \mathit{total}\ (\mathit{fmap}\ (\lambda(a,b) \to a+b)\ p)\ .\end{aligned}$$

This is not a straightforward fold, because the recursive call of *total* is not applied directly to a subterm—indeed, it cannot be so applied, because the subterm p of *Succ p* has type $\mathit{Perfect}\ (\mathbb{N},\mathbb{N})$ rather than $\mathit{Perfect}\ \mathbb{N}$.

Histomorphism Histomorphisms capture tabulation, as used in dynamic programming. For example, consider the unbounded knapsack problem: given an integer capacity c, and as many copies as needed of each of a collection of items (w_i, v_i) with *positive* integer weight w_i and value v_i, compute the maximum value that will fit in the knapsack. Thus, given capacity 15 and items $[(12,4),(1,2),(2,2),(1,1),(4,10)]$, the maximum value possible is 36, using three copies each of the second and fifth items.

The naive recursive solution takes exponential time (we suppose here that the maximum value of the empty list of candidate solutions is zero):

$$\begin{aligned}&\mathit{knapsack} :: [(\mathbb{N},\mathbb{R})] \to \mathbb{N} \to \mathbb{R}\\ &\mathit{knapsack}\ \mathit{wvs}\ c\\ &\quad = \mathit{maximum}\ [v + \mathit{knapsack}\ \mathit{wvs}\ (c{-}w) \mid (w,v) \leftarrow \mathit{wvs}, w \leqslant c]\ .\end{aligned}$$

However, by tabulating the results for each capacity in $0\,..\,c$, one can compute the answer in pseudo-polynomial time:

$$\begin{aligned}&\mathit{knapsack}\ \mathit{wvs}\ c = \mathit{table}\ !!\ c\ \textbf{where}\\ &\quad \mathit{table} = [\mathit{ks}\ i \mid i \leftarrow [0\,..\,c]]\\ &\quad \mathit{ks}\ i = \mathit{maximum}\ [v + \mathit{table}\ !!\ (i{-}w) \mid (w,v) \leftarrow \mathit{wvs}, w \leqslant i]\ .\end{aligned}$$

Lazy evaluation works out the data dependencies automatically; but each element of the table depends only on elements with lower indices, so even without lazy evaluation it suffices to fill the table in index order.

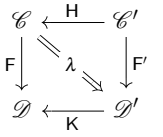
Now, the general question is whether the recursion equations above have unique solutions? The answer is yes for all of them. However, up to now the proofs involved two seemingly incompatible techniques: most of the examples can be identified as adjoint folds; some of them (in particular *knapsack*) are subsumed by recursion schemes from comonads. Before we show how to unify the two approaches, we first need to introduce a bit of theory.

3. Background

This paper assumes a basic knowledge of category theory, in that the reader should be familiar with the notions of functors, natural transformations, and product and functor categories. In this section we fix the notation and establish categorical concepts that will be used in the remainder of the paper. For the most part this material is standard and can safely be glossed over on an initial reading. An exception is perhaps the material on functor squares and conjugates, which will need particular attention.

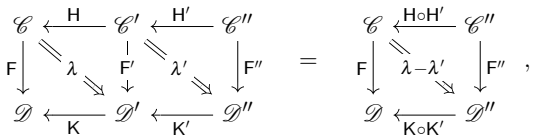
3.1 Functor Squares

A *functor square* consists of four functors and a natural transformation between them (to read off the type of λ, it might help to tilt your head 45° to the left when looking at this diagram):



$$\lambda : \mathsf{F} \circ \mathsf{H} \dot{\rightarrow} \mathsf{K} \circ \mathsf{F}' \ .$$

For brevity, we call λ a *distributive law*, even though the name is traditionally used for the special case in which opposite functors are comonads or monads. Functor squares can be horizontally (and also vertically, not shown below) composed:



where the horizontal composition $\lambda - \lambda'$ of the distributive laws λ and λ' is given by a combination of horizontal ($\circ$) and vertical ($\cdot$) composition of natural transformations ($\circ$ binds tighter than $\cdot$):

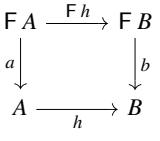
$$\lambda - \lambda' \ = \ \mathsf{K} \circ \lambda' \ \cdot \ \lambda \circ \mathsf{H}' \ .$$

This composition is associative, with $id_{\mathsf{F}} : \mathsf{F} \circ \mathsf{Id} \dot{\rightarrow} \mathsf{Id} \circ \mathsf{F}$ as its neutral element.

3.2 Algebras and Coalgebras

Algebras and coalgebras form the basis for the categorical description of structured recursion schemes.

Given an endofunctor $\mathsf{F} : \mathscr{C} \rightarrow \mathscr{C}$, an F-algebra is a pair (a, A), where $a : \mathsf{F}\ A \rightarrow A$ is an arrow and $A : \mathscr{C}$ is an object, which are known as the *action* and *carrier* of the algebra. (We deviate a little from the standard notation (A, a), in order to have a syntax that distinguishes algebras from coalgebras.) Since the action determines its carrier, it is often used by itself to refer to the F-algebra. An F-homomorphism between algebras (a, A) and (b, B) is an arrow $h : A \rightarrow B : \mathscr{C}$ such that $h \cdot a = b \cdot \mathsf{F}\ h$.



Clearly, F-homomorphisms compose and have an identity, so it follows that F-algebras and F-homomorphisms form a category, which we call F-**Alg**$(\mathscr{C})$. The initial object of this category, if it exists, is given by $(in, \mu\mathsf{F})$ and called the *initial* F*-algebra*. The initiality implies that to each F-algebra, (a, A), there exists a unique F-homomorphism, $(\!|a|\!) : (in, \mu\mathsf{F}) \rightarrow (a, A)$, called a *fold*. The algebra *in* is, in fact, an isomorphism, so $\mu\mathsf{F}$ is a fixed-point of F (the least fixed-point), a fact known as Lambek's lemma.

Example 3.1. The semantics of the inductive datatype *Tree* is given by the initial algebra $\mu\mathsf{Tree}$, where the so-called *base functor*

$$\textbf{data}\ \mathsf{Tree}\ \mathit{tree} = \mathsf{Empty} \mid \mathsf{Node}\ \mathit{tree}\ \mathbb{N}\ \mathit{tree}$$

abstracts away from the recursive occurrences of *Tree*. The Haskell rendering of the isomorphism *in*, the action of the initial algebra,

$$\begin{aligned} &\mathit{in} :: \mathsf{Tree}\ \mathit{Tree} \rightarrow \mathit{Tree} \\ &\mathit{in}\ (\mathsf{Empty}) \quad\ = \mathit{Empty} \\ &\mathit{in}\ (\mathsf{Node}\ l\ a\ r) = \mathit{Node}\ l\ a\ r \end{aligned}$$

amounts to a simple renaming of constructors. □

Dually, given an endofunctor $\mathsf{G} : \mathscr{C} \rightarrow \mathscr{C}$, a G-coalgebra is a pair (C, c), where $C : \mathscr{C}$ is the carrier and $c : C \rightarrow \mathsf{G}\ C$ is the action of the coalgebra. A G-homomorphism between coalgebras (C, c) and (D, d) is an arrow $h : C \rightarrow D : \mathscr{C}$ that satisfies $\mathsf{G}\ h \cdot c = d \cdot h$. Just as before, a category G-**Coalg**$(\mathscr{C})$ can be formed from G-coalgebras and G-homomorphisms. The final object of this category, if it exists, is given by $(\nu\mathsf{G}, \mathit{out})$ and called the *final* G*-coalgebra*.

The category F-**Alg**$(\mathscr{C})$ has more structure than $\mathscr{C}$. The forgetful or underlying functor $\mathsf{U}^{\mathsf{F}} : \mathsf{F}$-**Alg**$(\mathscr{C}) \rightarrow \mathscr{C}$ forgets about the additional structure: $\mathsf{U}^{\mathsf{F}}\ (a, A) = A$ and $\mathsf{U}^{\mathsf{F}}\ h = h$. An analogous functor can be defined for coalgebras: $\mathsf{U}_{\mathsf{G}} : \mathsf{G}$-**Coalg**$(\mathscr{C}) \rightarrow \mathscr{C}$.

Liftings and coliftings A functor $\bar{\mathsf{H}} : \mathsf{F}$-**Alg**$(\mathscr{C}) \rightarrow \mathsf{G}$-**Alg**$(\mathscr{D})$ is called a *lifting* of $\mathsf{H} : \mathscr{C} \rightarrow \mathscr{D}$ iff $\mathsf{H} \circ \mathsf{U}^{\mathsf{F}} = \mathsf{U}^{\mathsf{G}} \circ \bar{\mathsf{H}}$. Given a distributive law $\lambda : \mathsf{H} \circ \mathsf{F} \dot{\leftarrow} \mathsf{G} \circ \mathsf{H}$, we can define a lifting as follows:

$$\mathsf{H}^{\lambda}\ (a, A) = (\mathsf{H}\ a \cdot \lambda\ A, \mathsf{H}\ A) \ , \tag{3.1a}$$

$$\mathsf{H}^{\lambda}\ h = \mathsf{H}\ h \ . \tag{3.1b}$$

For liftings, the action on the carrier and on homomorphisms is fixed; the action on the algebra is determined by the distributive law. Liftings of the identity functor, that is, $\mathsf{H} = \mathsf{Id}$ and $\lambda = \alpha : \mathsf{F} \dot{\leftarrow} \mathsf{G}$, are often written as α-**Alg**$(\mathscr{C}) : \mathsf{F}$-**Alg**$(\mathscr{C}) \rightarrow \mathsf{G}$-**Alg**$(\mathscr{C})$. Liftings compose in an attractive way: $\mathsf{H}^{\lambda} \circ \mathsf{H}'^{\lambda'} = (\mathsf{H} \circ \mathsf{H}')^{\lambda - \lambda'}$.

Since we use the action of an algebra to refer to the algebra itself, we often abbreviate $\mathsf{H}\ a \cdot \lambda\ A$ by $\mathsf{H}^{\lambda}\ a$.

Dually, $\underline{\mathsf{H}} : \mathsf{F}$-**Coalg**$(\mathscr{C}) \rightarrow \mathsf{G}$-**Coalg**$(\mathscr{D})$ is a colifting of $\mathsf{H} : \mathscr{C} \rightarrow \mathscr{D}$ iff $\mathsf{U}_{\mathsf{G}} \circ \underline{\mathsf{H}} = \mathsf{H} \circ \mathsf{U}_{\mathsf{F}}$. Given $\lambda : \mathsf{H} \circ \mathsf{F} \dot{\rightarrow} \mathsf{G} \circ \mathsf{H}$ we can define a colifting as follows:

$$\mathsf{H}_{\lambda}\ (C, c) = (\mathsf{H}\ C, \lambda\ C \cdot \mathsf{H}\ c) \ , \tag{3.2a}$$

$$\mathsf{H}_{\lambda}\ h = \mathsf{H}\ h \ . \tag{3.2b}$$

3.3 Adjunctions

Adjunctions were introduced by Kan [14] and are so pervasive in the study of category theory that Mac Lane [17, p.vii] noted "Adjoint functors arise everywhere." Our work supports this view: adjunctions provide a unified framework for program transformation.

Given categories $\mathscr{C}, \mathscr{D}$, we say that functors $\mathsf{L} : \mathscr{C} \leftarrow \mathscr{D}$ and $\mathsf{R} : \mathscr{C} \rightarrow \mathscr{D}$ form an adjunction, written $\mathsf{L} \dashv \mathsf{R} : \mathscr{C} \rightharpoonup \mathscr{D}$ or

iff there is a bijection between the sets of arrows

$$\lfloor - \rfloor : \mathscr{C}(\mathsf{L}\ A, B) \cong \mathscr{D}(A, \mathsf{R}\ B) : \lceil - \rceil \ ,$$

that is natural both in A and B. We say that L is a *left adjoint* for R, and R a *right adjoint* for L; the isomorphism $\lfloor - \rfloor$ is called the *left adjunct*, and its inverse $\lceil - \rceil$ the *right adjunct*. The arrows $\lfloor f \rfloor$ and $\lceil g \rceil$ are also called the *transposes* of f and g.

That the adjuncts $\lfloor - \rfloor$ and $\lceil - \rceil$ are mutually inverse can be captured using an equivalence:

$$f = \lceil g \rceil \quad \Longleftrightarrow \quad \lfloor f \rfloor = g \ , \tag{3.3}$$

for all $f : \mathsf{L}\,A \to B : \mathscr{C}$ and $g : A \to \mathsf{R}\,B : \mathscr{D}$. The naturality properties of the adjuncts can be expressed as fusion laws.

$$\mathsf{R}\,k \cdot \lfloor f \rfloor \cdot h = \lfloor k \cdot f \cdot \mathsf{L}\,h \rfloor \tag{3.4a}$$

$$k \cdot \lceil g \rceil \cdot \mathsf{L}\,h = \lceil \mathsf{R}\,k \cdot g \cdot h \rceil \tag{3.4b}$$

These equations imply that the adjuncts are uniquely defined by their actions on the identity: $\mathsf{R}\,k \cdot \lfloor id \rfloor = \lfloor k \rfloor$ and $\lceil id \rceil \cdot \mathsf{L}\,h = \lceil h \rceil$. An alternative definition of adjunctions is based on the two natural transformations $\epsilon = \lceil id \rceil$ and $\eta = \lfloor id \rfloor$, which are called the *counit* $\epsilon : \mathsf{L} \circ \mathsf{R} \dot{\to} \mathsf{Id}$ and the *unit* $\eta : \mathsf{Id} \dot{\to} \mathsf{R} \circ \mathsf{L}$ of the adjunction. The equivalence (3.3) can also be framed in terms of the units:

$$f = \epsilon\,B \cdot \mathsf{L}\,g \iff \mathsf{R}\,f \cdot \eta\,A = g \ . \tag{3.5}$$

Adjunctions satisfy a wealth of properties. An important property is that adjoint functors are uniquely defined up to isomorphism: if $\mathsf{L}_1 \dashv \mathsf{R}_1$ and $\mathsf{L}_2 \dashv \mathsf{R}_2$, then

$$\mathsf{L}_2 \cong \mathsf{L}_1 \iff \mathsf{R}_1 \cong \mathsf{R}_2 \ . \tag{3.6}$$

This equivalence can be used as a reasoning principle: often one isomorphism is trivial and can be used to establish the other.

Left adjoints preserve initial objects, $\mathsf{L}\,0 \cong 0$. Dually, right adjoints preserve final objects, $\mathsf{R}\,1 \cong 1$. In general, left adjoints preserve colimits (LAPC) and right adjoints preserve limits (RAPL).

Example 3.2. Coproducts and products arise as left and right adjoints $(+) \dashv \Delta \dashv (\times)$ of the diagonal functor $\Delta : \mathscr{C} \to \mathscr{C} \times \mathscr{C}$ defined by $\Delta\,A = (A,A)$ and $\Delta\,f = (f,f)$.

$$\mathscr{C} \underset{\Delta}{\overset{(+)}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \times \mathscr{C} \qquad \mathscr{C} \times \mathscr{C} \underset{(\times)}{\overset{\Delta}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C}$$

The bijections express that pairs of arrows with the same source (respectively, target) are in one-to-one correspondence with arrows to a product (respectively, from a coproduct). In the case of products, the left adjunct $\lfloor (f_1, f_2) \rfloor = f_1 \vartriangle f_2$ is known as the 'split' combinator, and the counit $\epsilon = (outl, outr)$ arises from the projections. □

Example 3.3. Perhaps the best-known example of an adjunction is currying: a function of two arguments can be treated as a function of the first argument whose values are functions of the second.

$$\mathscr{C} \underset{(-)^P}{\overset{-\times P}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C}$$

The right adjoint of pairing with P is the exponential from P. □

Example 3.4. For a signature expressed as a functor F, the terms involving variables of type A constitute the free algebra $\mathsf{Free}^{\mathsf{F}}\,A$ on A. The functor $\mathsf{Free}^{\mathsf{F}} : \mathscr{C} \to \mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C})$ arises as the left adjoint of the forgetful functor U^{F}. Dually, cofree G-coalgebras arise as the right adjoint of U_{G}.

$$\mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C}) \underset{\mathsf{U}^{\mathsf{F}}}{\overset{\mathsf{Free}^{\mathsf{F}}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \qquad \mathscr{C} \underset{\mathsf{Cofree}_{\mathsf{G}}}{\overset{\mathsf{U}_{\mathsf{G}}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathsf{G}\text{-}\mathbf{Coalg}(\mathscr{C})$$

The first bijection expresses that the compositional evaluation of a term is uniquely determined by the action on variables. Initial algebras and final coalgebras arise as special cases (LAPC and RAPL): $(in, \mu\mathsf{F}) \cong \mathsf{Free}^{\mathsf{F}}\,0$ (closed terms as open terms where the variables are drawn from 0) and $(\nu\mathsf{G}, out) \cong \mathsf{Cofree}_{\mathsf{G}}\,1$. □

Adjunctions can be lifted to functor categories: $\mathsf{L} \dashv \mathsf{R}$ implies both $\mathsf{L} \circ - \dashv \mathsf{R} \circ -$ and $- \circ \mathsf{R} \dashv - \circ \mathsf{L}$. The latter adjunctions capture the following bijections between natural transformations:

$$\mathscr{C}^{\mathscr{X}}(\mathsf{L} \circ \mathsf{F}, \mathsf{G}) \cong \mathscr{D}^{\mathscr{X}}(\mathsf{F}, \mathsf{R} \circ \mathsf{G}) \ , \tag{3.7a}$$

$$\mathscr{X}^{\mathscr{C}}(\mathsf{F} \circ \mathsf{R}, \mathsf{G}) \cong \mathscr{X}^{\mathscr{D}}(\mathsf{F}, \mathsf{G} \circ \mathsf{L}) \ . \tag{3.7b}$$

Conjugates Next we introduce a concept that will be at the heart of our framework. Just as natural transformations relate functors, conjugates relate adjoint pairs of functors. Given the adjunctions $\mathsf{L} \dashv \mathsf{R} : \mathscr{C} \rightharpoonup \mathscr{D}$ and $\mathsf{L}' \dashv \mathsf{R}' : \mathscr{C}' \rightharpoonup \mathscr{D}'$, and functors $\mathsf{H} : \mathscr{C} \to \mathscr{C}'$ and $\mathsf{K} : \mathscr{D} \to \mathscr{D}'$, the distributive laws $\sigma : \mathsf{L}' \circ \mathsf{K} \dot{\to} \mathsf{H} \circ \mathsf{L}$ and $\tau : \mathsf{K} \circ \mathsf{R} \dot{\to} \mathsf{R}' \circ \mathsf{H}$ are *conjugates*, written $\sigma \dashv \tau$, if one of the following conditions holds

$$\lfloor \mathsf{H}\,f \cdot \sigma\,A \rfloor' = \tau\,B \cdot \mathsf{K} \lfloor f \rfloor \ , \tag{3.8a}$$

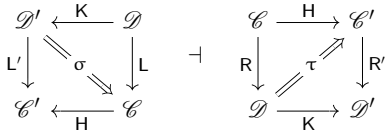
$$\mathsf{H} \lceil g \rceil \cdot \sigma\,A = \lceil \tau\,B \cdot \mathsf{K}\,g \rceil' \ , \tag{3.8b}$$

for all $f : \mathsf{L}\,A \to B : \mathscr{C}$ and $g : A \to \mathsf{R}\,B : \mathscr{D}$. The equivalence of the two conditions is a consequence of (3.7). In fact, each natural transformation uniquely determines the other:

$$\sigma\,A = \lceil \tau\,(\mathsf{L}\,A) \cdot \mathsf{K}\,(\eta\,A) \rceil' \ , \tag{3.9a}$$

$$\tau\,B = \lfloor \mathsf{H}\,(\epsilon\,B) \cdot \sigma\,(\mathsf{R}\,B) \rfloor' \ . \tag{3.9b}$$

We obtain two distributive laws for the price of one; this fact will be used a lot. The following diagrams record the types.



(As an aside, the data—the functors H and K and the laws σ and τ—are also called an *adjoint square*, a pair of functor squares, from $\mathsf{L} \dashv \mathsf{R}$ to $\mathsf{L}' \dashv \mathsf{R}'$. Above, we have taken the first steps towards defining the *double* category of adjoint squares [24].)

Example 3.5. A lifting $\bar{\mathsf{H}}$ provides an important example of a conjugate between categories of algebras where the second transformation $\tau : \mathsf{H} \circ \mathsf{U}^{\mathsf{F}} = \mathsf{U}^{\mathsf{G}} \circ \bar{\mathsf{H}}$ is manifestly the identity. □

4. Warm-up: An Easy Instance of Adjoint Folds

Before we introduce the unified framework, it is instructive to walk through a specific instance. In Section 2 we have discussed that functions defined by mutual recursion, *mutumorphisms*, are not simple folds. They are, however, in one-to-one correspondence with folds. Mutumorphisms are captured by the following scheme:

$$x_1 \cdot in = b_1 \cdot \mathsf{D}\,(x_1 \vartriangle x_2) \ \text{ and } \ x_2 \cdot in = b_2 \cdot \mathsf{D}\,(x_1 \vartriangle x_2) \ .$$

The split combinator makes the results of both recursive calls available to the 'algebras' $b_i : \mathsf{D}\,(B_1 \times B_2) \to B_i$. Think of $x_i : \mu\mathsf{D} \to B_i$ as unknowns; we aim to show that they are uniquely determined by the two equations. We proceed in two steps:

First, we abstract away from the initial algebra $(in, \mu\mathsf{D})$, generalising to an arbitrary D-algebra (a, A), and turn the two equations into a form we can work with. Product categories provide a natural setting, simply because we have two equations. (Recall that split $\vartriangle$ is the left adjunct of $\Delta \dashv (\times)$, see Example 3.2.)

$$\begin{array}{ll} & x_1 \cdot a = b_1 \cdot \mathsf{D}\,(x_1 \vartriangle x_2) \ \text{ and } \ x_2 \cdot a = b_2 \cdot \mathsf{D}\,(x_1 \vartriangle x_2) \\ \iff & \{\ \text{product category } \mathscr{C} \times \mathscr{C}\ \} \\ & (x_1, x_2) \cdot (a, a) = (b_1, b_2) \cdot (\mathsf{D}\,(x_1 \vartriangle x_2), \mathsf{D}\,(x_1 \vartriangle x_2)) \\ \iff & \{\ \text{definition of } \Delta \text{ and definition of } \lfloor - \rfloor\ \} \\ & (x_1, x_2) \cdot \Delta\,a = (b_1, b_2) \cdot \Delta\,(\mathsf{D}\,\lfloor (x_1, x_2) \rfloor) \\ \iff & \{\ \text{set } x := (x_1, x_2) \text{ and } b := (b_1, b_2)\ \} \\ & x \cdot \Delta\,a = b \cdot \Delta\,(\mathsf{D}\,\lfloor x \rfloor) \end{array}$$

We obtain a single equation, where the algebra a is wrapped in a left adjoint. From here, a short calculation demonstrates that the

transpose of x is a homomorphism:

$$\begin{array}{ll} & x \cdot \Delta\, a = b \cdot \Delta\,(\mathsf{D}\,\lfloor x \rfloor) \;\;:\;\; \Delta\,(\mathsf{D}\,A) \to B \\ \iff & \quad \{\ \lfloor - \rfloor \text{ and } \lceil - \rceil \text{ are isomorphisms (3.3)}\ \} \\ & \lfloor x \cdot \Delta\, a \rfloor = \lfloor b \cdot \Delta\,(\mathsf{D}\,\lfloor x \rfloor) \rfloor \\ \iff & \quad \{\ \lfloor - \rfloor \text{ is natural (3.4a)}\ \} \\ & \lfloor x \rfloor \cdot a = \lfloor b \rfloor \cdot \mathsf{D}\,\lfloor x \rfloor \;\;:\;\; \mathsf{D}\,A \to (\times)\,B \;\;. \end{array}$$

Thus, $\lfloor x \rfloor$ is a D-homomorphism, and so x is the transpose of a D-homomorphism. Furthermore, b is the transpose of a D-algebra—this is an important observation. Let us record the correspondence (note that B is an object of $\mathscr{C} \times \mathscr{C}$, that is, a pair of objects in $\mathscr{C}$, and recall that $\lfloor x \rfloor = x_1 \vartriangle x_2$).

$$\begin{array}{ccc} \Delta\,(\mathsf{D}\,A) & \xrightarrow{\Delta\,(\mathsf{D}\,\lfloor x \rfloor)} & \Delta\,(\mathsf{D}\,((\times)\,B)) \\ {\scriptstyle \Delta\, a}\downarrow & & \downarrow{\scriptstyle b} \\ \Delta\, A & \xrightarrow[x]{} & B \end{array} \quad \Longleftrightarrow \quad \begin{array}{ccc} \mathsf{D}\,A & \xrightarrow{\mathsf{D}\,\lfloor x \rfloor} & \mathsf{D}\,((\times)\,B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle \lfloor b \rfloor} \\ A & \xrightarrow[\lfloor x \rfloor]{} & (\times)\,B \end{array}$$

Second, we instantiate (a, A) to the initial algebra $(in, \mu\mathsf{D})$. The solution of the original pair of equations is then given by

$$x_1 \vartriangle x_2 = (\!|\, b_1 \vartriangle b_2 \,|\!) \;\;,$$

which is Fokkinga's mutu-CHARN law [7].

Several special cases are worth singling out. If x_2 does not depend on x_1, we obtain *zygomorphisms* (ie $b_2 := b \cdot \mathsf{D}\ outr$ and consequently $x_2 := (\!|\, b \,|\!)$). Further, when h_2 is the identity, the zygomorphism specialises to a paramorphism (ie $b_2 := in \cdot \mathsf{D}\ outr$ and consequently $x_2 := (\!|\, in \,|\!) = id$). Pushing this to the extreme, if we have two independent homomorphisms (ie $b_1 := b_1 \cdot \mathsf{D}\ outl$ and $b_2 := b_2 \cdot \mathsf{D}\ outr$ and consequently $x_1 = (\!|\, b_1 \,|\!)$ and $x_2 = (\!|\, b_2 \,|\!)$), we derive the *banana-split law* [3], an important program optimisation that replaces a double tree traversal by a single one.

$$(\!|\, b_1 \,|\!) \vartriangle (\!|\, b_2 \,|\!) = (\!|\, b_1 \cdot \mathsf{D}\ outl \vartriangle b_2 \cdot \mathsf{D}\ outr \,|\!) \tag{4.1}$$

The law can also be justified in a different way: $(\!|\, b_1 \,|\!) \vartriangle (\!|\, b_2 \,|\!)$ is the unique homomorphism *to a product algebra*:

$$(b_1, B_1) \times (b_2, B_2) = (b_1 \cdot \mathsf{D}\ outl \vartriangle b_2 \cdot \mathsf{D}\ outr, B_1 \times B_2) \;\;.$$

We shall see later that this is not just a lucky coincidence.

5. A Unified Framework for Recursion Schemes

This section introduces the promised unifying theory for recursion schemes. As noted in the introduction, the unifying concept, called *generalised iteration* in [20] and *adjoint fold* in [11], is not new. (So no new theory is needed, which is good.) What is novel is the presentation, which makes essential use of conjugate pairs of distributive laws and liftings, rendering the proofs concise and elegant. However, since the concepts from category theory are perhaps somewhat remote from the daily practice of the programmer in the cubicle, we first take a short detour, which hopefully helps to connect the abstract concepts to concrete programs.

5.1 Detour: Mendler-style Folds

Mendler-style folds [23, 28] arise from taking a logical (specifically, second-order simply-typed lambda calculus) rather than an algebraic approach to inductive datatypes. As such, they provide a smooth transition path from explicit recursion to the use of recursion schemes. To illustrate, the semantics of *depth* is roughly the fixed-point of the so-called *base function* depth

$$\begin{array}{l} \mathsf{depth}\ depth\ (\mathsf{Empty}) \quad\;\; = 0 \\ \mathsf{depth}\ depth\ (\mathsf{Node}\ l\ a\ r) = 1 + (depth\ l\ `max`\ depth\ r) \;\;, \end{array}$$

which abstracts away from the recursive calls. There is an additional twist: we have replaced the *Tree* constructors by the corresponding Tree constructors, which results in a rank-1 type:

$$\mathsf{depth} :: \forall tree\ .\ (tree \to \mathbb{N}) \to (\mathsf{Tree}\ tree \to \mathbb{N}) \;\;.$$

The polymorphic type guarantees that the original recursion equation, $depth \cdot in = \mathsf{depth}\ depth$ has a *unique* solution. (Because of the occurrence of *in*, we said 'roughly the fixed-point'.)

Translated into category theory, *Mendler-style folds* are solutions in an unknown $x : \mu\mathsf{D} \to B$ to recursion equations of the form

$$x \cdot in = \Psi\,(\mu\mathsf{D})\,x \;\;,$$

where the base function Ψ is a natural transformation of type $\mathscr{C}(-, B) \dot{\to} \mathscr{C}(\mathsf{D}\,-, B)$. Very briefly, the Yoneda lemma [17] shows that the space of base functions such as Ψ is isomorphic to the space of D-algebras. Thus, Mendler-style folds are in one-to-one correspondence with standard folds of the form

$$x \cdot in = b \cdot \mathsf{D}\,x \;\;.$$

Conversely, a standard fold is a Mendler-style fold, as the right-hand side as a function in x satisfies the naturality requirement.

5.2 Detour: Mendler-style Adjoint Folds

We have noted in Section 2 that many functions do not quite fit the pattern of simple folds: *depths*, for instance, uses an accumulating parameter. However, to provide a precise semantics we can take a similar approach as in the previous section. We define a base function that additionally replaces the *Tree* constructors on the left-hand side (and only those) by the corresponding Tree constructors.

$$\begin{array}{l} \mathsf{depths} :: \forall tree\ .\ ((tree, \mathbb{N}) \to Tree) \to ((\mathsf{Tree}\ tree, \mathbb{N}) \to Tree) \\ \mathsf{depths}\ depths\ (\mathsf{Empty}, \quad\;\; n) = Empty \\ \mathsf{depths}\ depths\ (\mathsf{Node}\ l\ a\ r, n) \\ \quad = Node\ (depths\ (l, n+1))\ n\ (depths\ (r, n+1)) \end{array}$$

The type of the base function is similar to what we had before, except that *tree* and Tree *tree* are wrapped in a left adjoint: $(-, \mathbb{N})$ or, categorically speaking, $- \times \mathbb{N}$. Nonetheless, one can show that $\mathsf{depths} \cdot (in \times \mathbb{N}) = \mathsf{depths}\ depths$ has a *unique* solution.

This motivates the following generalisation of Mendler-style folds. Given an adjunction $\mathsf{L} \dashv \mathsf{R}$, an *Mendler-style adjoint fold* $x : \mathsf{L}\,(\mu\mathsf{D}) \to B$ is the unique solution to the recursion equation

$$x \cdot \mathsf{L}\ in = \Psi\,(\mu\mathsf{D})\,x \;\;, \tag{5.1}$$

where the base function Ψ is now a natural transformation of type $\mathscr{C}(\mathsf{L}\,-, B) \dot{\to} \mathscr{C}(\mathsf{L}\,(\mathsf{D}\,-), B)$.

The main difficulty in translating the examples of Section 2 into adjoint folds is to identify the left adjoint. For some examples this is obvious, eg for *depths* we use the curry adjunction $- \times \mathbb{N} \dashv (-)^{\mathbb{N}}$; for others it is less obvious, eg for *total* the left adjoint is type application (applying a functor to a constant object), which has a right adjoint under some mild conditions [11].

5.3 Adjoint Folds

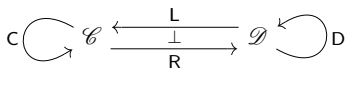
Standard folds are restricted to the case that the control structure of a function ever follows the structure of its input data. Mendler-style adjoint folds loosen this tight coupling. The control structure is given implicitly through the adjunction, but it can also be made explicit by introducing a 'control functor'.

Definition 5.1 (Adjoint recursion equation)**.** Given an adjunction $\mathsf{L} \dashv \mathsf{R} : \mathscr{C} \rightharpoonup \mathscr{D}$, functors $\mathsf{C} : \mathscr{C} \to \mathscr{C}$ and $\mathsf{D} : \mathscr{D} \to \mathscr{D}$, and a distributive law $\sigma : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L}$, an adjoint recursion equation in the unknown $x : \mathsf{L}\,(\mu\mathsf{D}) \to B$ has the form

$$x \cdot \mathsf{L}\ in = b \cdot \mathsf{C}\,x \cdot \sigma\,(\mu\mathsf{D}) \;\;, \tag{5.2}$$

where $b : \mathsf{C}\,B \to B$.

The functor C is called control functor because it governs the recursive call structure. The diagram below displays the functors involved (D as in data functor, C as in control functor).



(5.3)

The distributive law $\sigma : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L}$ serves as an impedance matcher relating data and control functors.

Adjoint folds arise as unique solutions of adjoint recursion equations—we postpone the proof of uniqueness until Section 5.4. As in the vanilla case, Mendler-style adjoint folds (5.1) and adjoint folds (5.2) are interchangeable. Every adjoint fold is a Mendler-style one, as the right-hand side of (5.2) as a function in x satisfies the naturality requirement. The other direction is more interesting:

Given a base function $\Psi : \mathscr{C}(\mathsf{L}\,-,B) \dot{\to} \mathscr{C}(\mathsf{L}\,(\mathsf{D}\,-),B)$, we have to construct a control functor C, a distributive law $\sigma : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L}$ and a C-algebra $b : \mathsf{C}\,B \to B$. The first two pieces of data are induced by the adjunction: a canonical choice for the control structure is $\mathsf{C} = \mathsf{L} \circ \mathsf{D} \circ \mathsf{R}$—we simply go round in a loop (5.3). Using this definition, the type of σ expands to $\mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{L} \circ \mathsf{D} \circ \mathsf{R} \circ \mathsf{L}$, which suggests defining $\sigma = \mathsf{L} \circ \mathsf{D} \circ \eta$. Finally, the C-algebra is derived from the base function: $b = \Psi\,(\mathsf{R}\,B)\,(\epsilon\,B) : \mathsf{L}\,(\mathsf{D}\,(\mathsf{R}\,B)) \to B$. For the proof of $\Psi\,X\,x = b \cdot \mathsf{C}\,x \cdot \sigma\,X$ we refer to the original paper [11].

Recursive Haskell programs are easily framed as Mendler-style adjoint folds (5.1). Adjoint folds (5.2) are, however, preferable for the theoretical development as they avoid sophistications such as natural transformations between hom-functors.

5.4 Transposed Homomorphisms

To show that (5.2) has a unique solution, we proceed in two steps, following the pattern set out in Section 4.

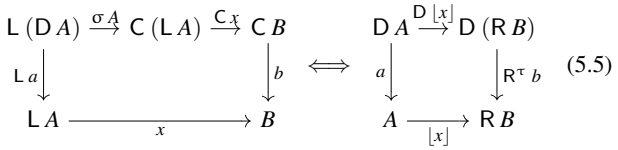
First, we abstract away from the initial algebra $(in, \mu\mathsf{D})$, generalising to an arbitrary D-algebra (a,A), and establish a bijection between arrows $x : \mathsf{L}\,A \to B$ satisfying

$$x \cdot \mathsf{L}\,a = b \cdot \mathsf{C}\,x \cdot \sigma\,A \ , \tag{5.4}$$

and D-algebra homomorphisms. The central step in the calculation below is the penultimate one, which replaces the distributive law $\sigma : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L}$ by its conjugate $\tau : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C}$, effectively shifting the recursive call to the right.

$$\begin{array}{ll}
& x \cdot \mathsf{L}\,a = b \cdot \mathsf{C}\,x \cdot \sigma\,A \ : \ \mathsf{L}\,(\mathsf{D}\,A) \to B \\
\iff & \{\ \lfloor - \rfloor \text{ and } \lceil - \rceil \text{ are isomorphisms (3.3)}\ \} \\
& \lfloor x \cdot \mathsf{L}\,a \rfloor = \lfloor b \cdot \mathsf{C}\,x \cdot \sigma\,A \rfloor \\
\iff & \{\ \lfloor - \rfloor \text{ is natural (3.4a)}\ \} \\
& \lfloor x \rfloor \cdot a = \mathsf{R}\,b \cdot \lfloor \mathsf{C}\,x \cdot \sigma\,A \rfloor \\
\iff & \{\ \sigma \dashv \tau \text{ conjugates (3.8a)}\ \} \\
& \lfloor x \rfloor \cdot a = \mathsf{R}\,b \cdot \tau\,B \cdot \mathsf{D}\,\lfloor x \rfloor \\
\iff & \{\ \text{definition of lifting (3.1a)}\ \} \\
& \lfloor x \rfloor \cdot a = \mathsf{R}^{\tau}\,b \cdot \mathsf{D}\,\lfloor x \rfloor \ : \ \mathsf{D}\,A \to \mathsf{R}\,B
\end{array}$$

Voilà: the transpose $\lfloor x \rfloor : (a,A) \to \mathsf{R}^{\tau}\,(b,B)$ is a D-homomorphism between a and a lifting of b. To fix some terminology, we call x a *transposed homomorphism*, or *traho* for short.



(5.5)

Second, if we instantiate (a,A) to the initial algebra $(in, \mu\mathsf{D})$, we obtain the following

Theorem 5.2 (Adjoint folds)**.** *The adjoint recursion equation* (5.2) *has the unique solution* $x = \lceil (\!| \mathsf{R}^{\tau}\,b |\!) \rceil$*, where* $\tau : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C}$ *is the conjugate of* σ*. The arrow* x *is called an* adjoint fold*.*

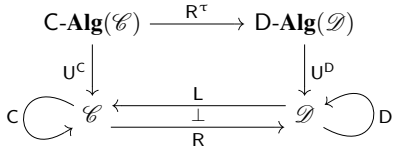
Proof. This is an immediate consequence of initiality.

$$\begin{array}{ll}
& x \cdot \mathsf{L}\,in = b \cdot \mathsf{C}\,x \cdot \sigma\,(\mu\mathsf{D}) \\
\iff & \{\ \text{see above}\ \} \\
& \lfloor x \rfloor \cdot in = \mathsf{R}^{\tau}\,b \cdot \mathsf{D}\,\lfloor x \rfloor \\
\iff & \{\ (in, \mu\mathsf{D}) \text{ initial}\ \} \\
& \lfloor x \rfloor = (\!| \mathsf{R}^{\tau}\,b |\!) \\
\iff & \{\ \lfloor - \rfloor \text{ and } \lceil - \rceil \text{ are isomorphisms (3.3)}\ \} \\
& x = \lceil (\!| \mathsf{R}^{\tau}\,b |\!) \rceil
\end{array}$$

□

So an adjoint fold is a traho from the initial algebra. Using the bijection (5.5) we can easily generalise from initial to free algebras. Then $\lfloor x \rfloor$ can be seen as evaluating a first-order term, and is uniquely determined by an evaluation function for variables.

There is an interesting observation to be made. Adjoint folds arise out of a situation that is not symmetric. The distributive law τ allows us to lift the right adjoint R to categories of algebras:

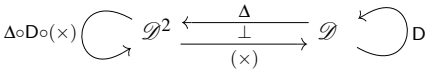


(5.6)

$$\sigma : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L} \ \dashv \ \tau : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C} \ .$$

Alas, we cannot lift the left adjoint L with the data at hand: a lifting of L requires a distributive law of type $\mathsf{C} \circ \mathsf{L} \dot{\to} \mathsf{L} \circ \mathsf{D}$. The asymmetry can be traced back to the definition of algebras. Consider the type of an action, $a : \mathsf{D}\,A \to A$; the base functor D only appears to the left of the arrow, in a contravariant position. Symmetry can be restored if σ is an isomorphism, an important special case, which we explore in the Section 5.5. But first, let us look at an example.

Example 5.3. Mutumorphisms are an instance of adjoint folds where the control functor is the canonical one and $\sigma = \Delta \circ \mathsf{D} \circ \eta$.



The conjugate of σ is $\tau = \eta \circ \mathsf{D} \circ (\times)$ (3.9b) and thus

$$(\times)^{\tau}\,(b_1, b_2) = b_1 \vartriangle b_2 \ .$$

Note that the lifted product functor is just the left adjunct. □

When we discussed adjoint folds (Section 5.3), we introduced the canonical control functor $\mathsf{C} = \mathsf{L} \circ \mathsf{D} \circ \mathsf{R}$. For this case the development above can be simplified. The functor C comes equipped with a canonical pair of distributive laws:

$$\sigma = \mathsf{L} \circ \mathsf{D} \circ \eta : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L} \ \dashv \ \tau = \eta \circ \mathsf{D} \circ \mathsf{R} : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C} \ .$$

The proof of uniqueness then boils down to a two-stepper (this is the proof of Section 4, more abstractly):

$$\begin{array}{ll}
& x \cdot \mathsf{L}\,a = b \cdot \mathsf{L}\,(\mathsf{D}\,\lfloor x \rfloor) \ : \ \mathsf{L}\,(\mathsf{D}\,A) \to B \\
\iff & \{\ \lfloor - \rfloor \text{ and } \lceil - \rceil \text{ are isomorphisms (3.3)}\ \} \\
& \lfloor x \cdot \mathsf{L}\,a \rfloor = \lfloor b \cdot \mathsf{L}\,(\mathsf{D}\,\lfloor x \rfloor) \rfloor \\
\iff & \{\ \lfloor - \rfloor \text{ is natural (3.4a)}\ \} \\
& \lfloor x \rfloor \cdot a = \lfloor b \rfloor \cdot \mathsf{D}\,\lfloor x \rfloor \ : \ \mathsf{D}\,A \to \mathsf{R}\,B \ .
\end{array}$$

This is indeed an instance of the previous development: some easy calculations show that $\mathsf{R}^\tau\ b = \lfloor b \rfloor$ and $\mathsf{L}\ (\mathsf{D}\ \lfloor x \rfloor) = \mathsf{C}\ x \cdot \sigma\ A$.

$$\begin{array}{ccc} \mathsf{L}\ (\mathsf{D}\ A) & \xrightarrow{\mathsf{L}\ (\mathsf{D}\ \lfloor x \rfloor)} & \mathsf{L}\ (\mathsf{D}\ (\mathsf{R}\ B)) \\ {\scriptstyle \mathsf{L}\ a}\downarrow & & \downarrow{\scriptstyle b} \\ \mathsf{L}\ A & \xrightarrow[x]{} & B \end{array} \quad \Longleftrightarrow \quad \begin{array}{ccc} \mathsf{D}\ A & \xrightarrow{\mathsf{D}\ \lfloor x \rfloor} & \mathsf{D}\ (\mathsf{R}\ B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle \lfloor b \rfloor} \\ A & \xrightarrow[\lfloor x \rfloor]{} & \mathsf{R}\ B \end{array} \tag{5.7}$$

Now, if (a,A) is initial, then $x = \lceil\!(\lfloor b \rfloor)\!\rceil$. We further explore the canonical control functor in Section 5.6.

5.5 Type Fusion

Let us now assume that the distributive law σ is an isomorphism. Then we can continue the first calculation of Section 5.4 'in the opposite direction'. We start with (5.4) and reason

$$\begin{array}{ll} & x \cdot \mathsf{L}\ a = b \cdot \mathsf{C}\ x \cdot \sigma\ A \ :\ \mathsf{L}\ (\mathsf{D}\ A) \to B \\ \Longleftrightarrow & \{\ \sigma \text{ is an isomorphism } \} \\ & x \cdot \mathsf{L}\ a \cdot \sigma^\circ\ A = b \cdot \mathsf{C}\ x \\ \Longleftrightarrow & \{\ \text{definition of lifting (3.1a)}\ \} \\ & x \cdot \mathsf{L}^{\sigma^\circ}\ a = b \cdot \mathsf{C}\ x \ :\ \mathsf{C}\ (\mathsf{L}\ A) \to B\ . \end{array}$$

Overall, we have established the following one-to-one correspondence between algebra homomorphisms.

$$\begin{array}{ccc} \mathsf{C}\ (\mathsf{L}\ A) & \xrightarrow{\mathsf{C}\ x} & \mathsf{C}\ B \\ {\scriptstyle \mathsf{L}^{\sigma^\circ}\ a}\downarrow & & \downarrow{\scriptstyle b} \\ \mathsf{L}\ A & \xrightarrow[x]{} & B \end{array} \quad \overset{\sigma \text{ iso}}{\Longleftrightarrow} \quad \begin{array}{ccc} \mathsf{D}\ A & \xrightarrow{\mathsf{D}\ \lfloor x \rfloor} & \mathsf{D}\ (\mathsf{R}\ B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle \mathsf{R}^\tau\ b} \\ A & \xrightarrow[\lfloor x \rfloor]{} & \mathsf{R}\ B \end{array}$$

In other words, jointly with L we have lifted the entire adjunction $\mathsf{L} \dashv \mathsf{R}$ to an adjunction $\mathsf{L}^{\sigma^\circ} \dashv \mathsf{R}^\tau$ between categories of algebras.

$$\mathsf{C}\text{-}\mathbf{Alg}(\mathscr{C})(\mathsf{L}^{\sigma^\circ}\ (a,A),(b,B)) \cong \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D})((a,A),\mathsf{R}^\tau\ (b,B))$$

We arrive at a situation that is perfectly symmetric. Trahos appear at some intermediate stage, at the point where we apply the assumption that the distributive law σ is an isomorphism.

We can now complete (5.6) with the missing left adjoints.

$$\begin{array}{ccc} \mathsf{C}\text{-}\mathbf{Alg}(\mathscr{C}) & \underset{\mathsf{R}^\tau}{\overset{\mathsf{L}^{\sigma^\circ}}{\leftarrow \bot \rightarrow}} & \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D}) \\ {\scriptstyle \mathsf{Free}^\mathsf{C}}\uparrow \dashv \downarrow{\scriptstyle \mathsf{U}^\mathsf{C}} & \cong & {\scriptstyle \mathsf{Free}^\mathsf{D}}\uparrow \dashv \downarrow{\scriptstyle \mathsf{U}^\mathsf{D}} \\ \mathsf{C} \circlearrowleft\ \mathscr{C} & \underset{\mathsf{R}}{\overset{\mathsf{L}}{\leftarrow \bot \rightarrow}} & \mathscr{D}\ \circlearrowright \mathsf{D} \end{array} \tag{5.8}$$

$$\sigma : \mathsf{L} \circ \mathsf{D} \cong \mathsf{C} \circ \mathsf{L} \ \dashv\ \tau : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C}$$

Overall, we have four (!) adjunctions, which form a commuting square of adjunctions. The proof of this fact makes use of the hgih-level reasoning principle (3.6). If we instantiate (3.6) to the compositions of left and right adjoints (note that left adjoints are composed in the opposite order) we obtain:

$$\mathsf{Free}^\mathsf{C} \circ \mathsf{L} \cong \mathsf{L}^{\sigma^\circ} \circ \mathsf{Free}^\mathsf{D} \quad \Longleftrightarrow \quad \mathsf{U}^\mathsf{D} \circ \mathsf{R}^\tau \cong \mathsf{R} \circ \mathsf{U}^\mathsf{C}\ .$$

Since R^τ is a lifting, the isomorphism on the right is valid—indeed, it is even an equality. Consequently, the compositions of left adjoints are isomorphic, as well. We record the following

Theorem 5.4. *Let $\mathsf{L} \dashv \mathsf{R} : \mathscr{C} \rightharpoonup \mathscr{D}$ be an adjunction, and let $\mathsf{C} : \mathscr{C} \to \mathscr{C}$ and $\mathsf{D} : \mathscr{D} \to \mathscr{D}$ be functors.*

$$\mathsf{L} \circ \mathsf{D} \cong \mathsf{C} \circ \mathsf{L} \quad \Longrightarrow \quad \mathsf{L} \circ \mathsf{D}^* \cong \mathsf{C}^* \circ \mathsf{L}$$

Proof. Plugging in the definitions, $\mathsf{F}^* = \mathsf{U}^\mathsf{F} \circ \mathsf{Free}^\mathsf{F}$, we conclude

$$\mathsf{L} \circ \mathsf{U}^\mathsf{D} \circ \mathsf{Free}^\mathsf{D} = \mathsf{U}^\mathsf{C} \circ \mathsf{L}^{\sigma^\circ} \circ \mathsf{Free}^\mathsf{D} \cong \mathsf{U}^\mathsf{C} \circ \mathsf{Free}^\mathsf{C} \circ \mathsf{L}\ . \qquad \square$$

As a corollary (using $\mu\mathsf{F} \cong \mathsf{F}^*\ 0$ and $\mathsf{L}\ 0 \cong 0$) we obtain the fusion rule of Backhouse et al. [1]'s categorical fixed-point calculus.

Corollary 5.5 (Type fusion)**.**

$$\mathsf{L} \circ \mathsf{D} \cong \mathsf{C} \circ \mathsf{L} \quad \Longrightarrow \quad \mathsf{L}\ (\mu\mathsf{D}) \cong \mu\mathsf{C}$$

Example 5.6. The diagonal functor Δ satisfies a simple property: $\Delta \circ \mathsf{D} = \mathsf{D}^2 \circ \Delta$. Since Δ is a left adjoint, Corollary 5.5 implies

$$\Delta\ (\mu\mathsf{D}) \cong \mu\mathsf{D}^2\ .$$

The initial algebra of D^2, a functor over a product category, consists of two copies of $\mu\mathsf{D}$—we will later need this simple fact. The conjugate of the distributive law $id : \Delta \circ \mathsf{D} = \mathsf{D}^2 \circ \Delta$ is $\tau = \mathsf{D}\ outl \vartriangle \mathsf{D}\ outr$ (3.9b) and thus

$$(\times)^\tau\ (b_1,b_2) = b_1 \cdot \mathsf{D}\ outl \vartriangle b_2 \cdot \mathsf{D}\ outr\ .$$

Instantiating Diagram (5.8) we can see the global picture.

$$\begin{array}{ccc} \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D})^2 \cong \mathsf{D}^2\text{-}\mathbf{Alg}(\mathscr{D}^2) & \underset{(\times)^\tau}{\overset{\Delta^{id}}{\leftarrow \bot \rightarrow}} & \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D}) \\ {\scriptstyle \mathsf{U}^{\mathsf{D}^2}}\downarrow & & \downarrow{\scriptstyle \mathsf{U}^\mathsf{D}} \\ \mathsf{D}^2 \circlearrowleft\ \mathscr{D}^2 & \underset{(\times)}{\overset{\Delta}{\leftarrow \bot \rightarrow}} & \mathscr{D}\ \circlearrowright \mathsf{D} \end{array}$$

Since $\mathsf{D}^2\text{-}\mathbf{Alg}(\mathscr{D}^2) \cong \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D})^2$, we obtain that $(\times)^\tau$ modulo the isomorphism is the product functor for $\mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D})$, which gives us the entire infrastructure for products: *outl*, *outr* and $\vartriangle$. (This also provides us with another proof of the banana-split law (4.1)) $\square$

We have now encountered two control functors associated with the adjunction $\Delta \dashv (\times)$: the canonical one $\Delta \circ \mathsf{D} \circ (\times)$ and the 'perfect' control functor D^2. The next section relates the two.

5.6 Comparing Control Functors

Adjoint folds involve several pieces of data: an adjunction, an algebra, and a control functor equipped with a conjugate pair of distributive laws. The latter is perhaps the most mysterious, especially if we try to link the Haskell programs in Section 2 *directly* to the recursion scheme of adjoint folds (5.2). Mendler-style folds provide a stepping stone, suggesting that there is a canonical choice for the control functor: $\mathsf{C} = \mathsf{L} \circ \mathsf{D} \circ \mathsf{R}$ with

$$\sigma = \mathsf{L} \circ \mathsf{D} \circ \eta : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L} \ \dashv\ \tau = \eta \circ \mathsf{D} \circ \mathsf{R} : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C}\ .$$

We now justify the adjective 'canonical' for this choice: we show that every other control functor can be reduced to the canonical one. Assume that we have another control functor C' with

$$\sigma' : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C}' \circ \mathsf{L} \ \dashv\ \tau' : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C}'\ .$$

Using bijection (3.7b), the distributive law σ' gives rise to a natural transformation $\gamma : \mathsf{L} \circ \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{C}' = \mathsf{C} \dot{\to} \mathsf{C}'$, namely $\gamma = \mathsf{C} \circ \epsilon \cdot \sigma' \circ \mathsf{R}$. This natural transformation in turn induces the lifting $\gamma\text{-}\mathbf{Alg}(\mathscr{C})$, which maps C'-algebras to C-algebras. Since it is a lifting of the identity functor, $\gamma\text{-}\mathbf{Alg}(\mathscr{C})$ is faithful. Moreover, we have the following commutative diagrams of functors.

$$\begin{array}{ccc} \mathsf{C}\text{-}\mathbf{Alg}(\mathscr{C}) & \xrightarrow{\mathsf{R}^\tau} & \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D}) \\ {\scriptstyle \gamma\text{-}\mathbf{Alg}(\mathscr{C})}\uparrow & & \| \\ \mathsf{C}'\text{-}\mathbf{Alg}(\mathscr{C}) & \xrightarrow[\mathsf{R}^{\tau'}]{} & \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D}) \end{array} \tag{5.9}$$

We first note that γ relates $\sigma \dashv \tau$ and $\sigma' \dashv \tau'$ in the following way (the proofs are routine but uninstructive).

$$\sigma' = \gamma \circ \mathsf{L} \cdot \sigma \tag{5.10a}$$
$$\tau' = \mathsf{R} \circ \gamma \cdot \tau \tag{5.10b}$$

For the proof of (5.9) it suffices to concentrate on the algebras:

$$\mathsf{R}^{\tau}\ (\gamma\text{-}\mathbf{Alg}(\mathscr{C})\ a) = \mathsf{R}\ (a \cdot \gamma\, A) \cdot \tau\, A = \mathsf{R}\, a \cdot \tau'\, A = \mathsf{R}^{\tau'}\ a\ .$$

Furthermore, every traho can be translated into a traho that uses the canonical control functor:

$$\begin{array}{ll} & x \cdot \mathsf{L}\ a = b \cdot \mathsf{C}'\, x \cdot \sigma'\, A \\ \iff & \{\ (5.10a)\ \} \\ & x \cdot \mathsf{L}\ a = b \cdot \mathsf{C}'\, x \cdot \gamma\ (\mathsf{L}\, A) \cdot \sigma\, A \\ \iff & \{\ \gamma \text{ is natural and } x : \mathsf{L}\, A \to B\ \} \\ & x \cdot \mathsf{L}\ a = b \cdot \gamma\, B \cdot \mathsf{C}\, x \cdot \sigma\, A \\ \iff & \{\ \text{definition of lifting (3.1b)}\ \} \\ & x \cdot \mathsf{L}\ a = \gamma\text{-}\mathbf{Alg}(\mathscr{C})\ b \cdot \mathsf{C}\, x \cdot \sigma\, A\ . \end{array}$$

Example 5.7. In Section 4 we noted that the banana-split law (4.1) arises as an extreme case of mutumorphisms. Mutumorphisms are based on the canonical control functor $\mathsf{C} = \Delta \circ \mathsf{D} \circ (\times)$; banana-split employs the control functor D^2. The lifting $\gamma\text{-}\mathbf{Alg}(\mathscr{C}^2) : \mathsf{D}^2\text{-}\mathbf{Alg}(\mathscr{C}^2) \to \mathsf{C}\text{-}\mathbf{Alg}(\mathscr{C}^2)$ induced by $\gamma = (\mathsf{D}\ \mathit{outl}, \mathsf{D}\ \mathit{outr}) : \mathsf{C} \to \mathsf{D}^2$ serves as the adaptor, translating D^2- into C-algebras. □

6. Recursion Schemes from Comonads

Recursion schemes from comonads [30], rsfcs for short, form a general recursion principle that makes use of a comonad N to provide 'contextual information' to the algebra. Like adjoint folds, it is 'doubly generic': it is parametric in the datatype $\mu\mathsf{F}$, and in the comonad N. As a particularly nice example, histomorphisms, the Squiggol rendering of course-of-values recursion, employ the cofree comonad, which makes available the results of recursive calls on *all* subterms. (An even better choice is the cofree *recursive* comonad [29].) To this end it makes use of a coalgebra $\mathit{fan} : \mu\mathsf{F} \to \mathsf{N}\ (\mu\mathsf{F})$ that embeds a subterm in a context. For the cofree comonad, *fan* maps a term to the cotree of all subterms. The coalgebra can be defined generically in terms of a distributive law $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$, which is subject to certain conditions (6.6), detailed below.

Definition 6.1 (Comonadic recursion equation)**.** Given a functor F, a comonad $(\mathsf{N}, \epsilon, \delta)$, and a distributive law $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$, a comonadic recursion equation in the unknown $f : \mu\mathsf{F} \to B$ has the form

$$f \cdot \mathit{in} = b \cdot \mathsf{F}\ (\mathsf{N}\, f \cdot \mathit{fan})\ , \tag{6.1}$$

where $\mathit{fan} = (\!|\mathsf{N}\ \mathit{in} \cdot \lambda\ (\mu\mathsf{F})|\!) : \mu\mathsf{F} \to \mathsf{N}\ (\mu\mathsf{F})$ and $b : \mathsf{F}\ (\mathsf{N}\, B) \to B$.

The composition $\mathsf{N}\, f \cdot \mathit{fan}$ creates a context that makes the results of 'recursive calls' available to the algebra b, which is a context-sensitive algebra—an $(\mathsf{F} \circ \mathsf{N})$-algebra, rather than merely an F-algebra. Uustalu et al. [30] showed the following

Theorem 6.2 (Rsfcs)**.** *The comonadic recursion equation* (6.1) *has the unique solution* $f = \epsilon\, B \cdot (\!|\mathsf{N}\, b \cdot \lambda\ (\mathsf{N}\, B) \cdot \mathsf{F}\ (\delta\, B)|\!)$.

A couple of remarks are in order. The recursion scheme involves both algebras and coalgebras, and combines them in an interesting way. We noted above that *fan* is a coalgebra, but it is actually a bit more: it is a coalgebra *for the comonad* N. Furthermore, the algebra *in* and the coalgebra *fan* go hand-in-hand. They are related by the distributive law λ and form what is known as a λ-bialgebra, a combination of an algebra and a coalgebra with a common carrier.

We postpone our proof of Theorem 6.2 to Section 6.3, after we have provided the necessary background in the following sections, which can be skipped by those already familiar with the material.

6.1 Background: Eilenberg-Moore Categories

Comonads and monads Functional programmers have embraced monads, and to a lesser extent, comonads, to capture effectful and context-sensitive computations. A comonad is a functor $\mathsf{N} : \mathscr{C} \to \mathscr{C}$ equipped with natural transformations $\epsilon : \mathsf{N} \dot{\to} \mathsf{Id}$ (counit), that extracts a value from a context, and $\delta : \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{N}$ (comultiplication), that duplicates a context, such that the following laws hold:

$$\epsilon \circ \mathsf{N} \cdot \delta = \mathsf{N}\ , \tag{6.2a}$$
$$\mathsf{N} \circ \epsilon \cdot \delta = \mathsf{N}\ , \tag{6.2b}$$
$$\delta \circ \mathsf{N} \cdot \delta = \mathsf{N} \circ \delta \cdot \delta\ . \tag{6.2c}$$

The first two properties, the counit laws, state that duplicating a context and then discarding a duplicate is the same as doing nothing. The third property, the coassociative law, equates the two ways of duplicating a context twice. Monads (M, η, μ) are dual to comonads, with transformations $\eta : \mathsf{Id} \dot{\to} \mathsf{M}$ (unit) and $\mu : \mathsf{M} \circ \mathsf{M} \dot{\to} \mathsf{M}$ (multiplication) that obey dual properties.

Huber [13] discovered that an adjunction $(\epsilon, \mathsf{L} \dashv \mathsf{R}, \eta)$ induces a comonad $(\mathsf{L} \circ \mathsf{R}, \epsilon, \mathsf{L} \circ \eta \circ \mathsf{R})$ and a monad $(\mathsf{R} \circ \mathsf{L}, \eta, \mathsf{R} \circ \epsilon \circ \mathsf{L})$. For example, the adjunction $\mathsf{Free}^{\mathsf{F}} \dashv \mathsf{U}^{\mathsf{F}}$ induces the so-called *free monad* $\mathsf{F}^* = \mathsf{U}^{\mathsf{F}} \circ \mathsf{Free}^{\mathsf{F}}$, the carrier of the free F-algebra, representing first-order terms with variables. (The comonad that arises is less interesting.) Dually, the adjunction $\mathsf{U}_{\mathsf{G}} \dashv \mathsf{Cofree}_{\mathsf{G}}$ induces the *cofree comonad* $\mathsf{G}_\infty = \mathsf{U}_{\mathsf{G}} \circ \mathsf{Cofree}_{\mathsf{G}}$. This can be seen as the type of generalised streams of observations—'generalised' because the 'tail' is a G-structure of 'streams' rather than just a single one; we obtain streams for $\mathsf{G} = \mathsf{Id}$. (Now the monad is less interesting.)

Coalgebras for a comonad A coalgebra for a comonad N is an N-coalgebra (C, c) that respects ϵ and δ:

$$\epsilon\ C \cdot c = \mathit{id}_C\ , \tag{6.3a}$$
$$\delta\ C \cdot c = \mathsf{N}\, c \cdot c\ . \tag{6.3b}$$

If we first create a context and then focus, we obtain the original value. Creating a nested context is the same as first creating a context and then duplicating it. For example, the so-called cofree coalgebra $(\mathsf{N}\ C, \delta\ C)$ is respectful, which follows directly from (6.2b) and (6.2c). The second law (6.3b) also enjoys an alternative reading: c is a homomorphism of type $(C, c) \to (\mathsf{N}\ C, \delta\ C)$. This observation is at the heart of the Eilenberg-Moore construction, which we discuss below. Coalgebras that respect ϵ and δ, and coalgebra homomorphisms between them, form a category, known as the *(co)-Eilenberg-Moore category* and denoted $\mathscr{C}_{\mathsf{N}}$. Eilenberg-Moore categories generalise categories of coalgebras: $\mathsf{G}\text{-}\mathbf{Coalg}(\mathscr{C}) \cong \mathscr{C}_{\mathsf{N}}$ where $\mathsf{N} = \mathsf{G}_\infty$ is the cofree comonad.

Eilenberg-Moore construction As noted above, every adjunction generates a comonad. The converse is also true: every comonad N induces an adjunction that generates N—in fact, in two canonical ways. One construction was discovered by Kleisli [15], the other by Eilenberg and Moore [6]. We shall need the latter, which constructs a right adjoint to the forgetful functor $\mathsf{U}_{\mathsf{N}} : \mathscr{C}_{\mathsf{N}} \to \mathscr{C}$.

$$\mathscr{C} \underset{\mathsf{Cofree}_{\mathsf{N}}}{\overset{\mathsf{U}_{\mathsf{N}}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C}_{\mathsf{N}}$$

The functor $\mathsf{Cofree}_{\mathsf{N}}$ maps an object to the cofree coalgebra for N:

$$\mathsf{Cofree}_{\mathsf{N}}\ B = (\mathsf{N}\, B, \delta\, B)\ , \tag{6.4a}$$
$$\mathsf{Cofree}_{\mathsf{N}}\, f = \mathsf{N}\, f\ . \tag{6.4b}$$

The counit $\epsilon : \mathsf{U}_{\mathsf{N}} \circ \mathsf{Cofree}_{\mathsf{N}} \dot{\to} \mathsf{Id}$ of the adjunction $\mathsf{U}_{\mathsf{N}} \dashv \mathsf{Cofree}_{\mathsf{N}}$ is the counit of N; the unit $\eta : \mathsf{Id} \dot{\to} \mathsf{Cofree}_{\mathsf{N}} \circ \mathsf{U}_{\mathsf{N}}$ defined $\eta\ (C, c) = c$ extracts the action of a coalgebra, which is an N-coalgebra homomorphism of type $(C, c) \to (\mathsf{N}\ C, \delta\ C)$ (6.3b). The bijection framed

in terms of the units reads:

$$f = \epsilon\ B \cdot \mathsf{U}_\mathsf{N}\ h \quad\Longleftrightarrow\quad \mathsf{Cofree}_\mathsf{N}\ f \cdot \eta\ (C,c) = h\ ,$$

for all $f\colon \mathsf{U}_\mathsf{N}\ (C,c) \to B$ and $h : (C,c) \to \mathsf{Cofree}_\mathsf{N}\ B$. The adjunction $\mathsf{U}_\mathsf{N} \dashv \mathsf{Cofree}_\mathsf{N}$ indeed generates the comonad N: we have $\mathsf{U}_\mathsf{N} \circ \mathsf{Cofree}_\mathsf{N} = \mathsf{N}$ and $\delta = \mathsf{U}_\mathsf{N} \circ \eta \circ \mathsf{Cofree}_\mathsf{N}$. Since U_N is faithful, we can simplify the bijection slightly:

$$f = \epsilon\ B \cdot h \quad\Longleftrightarrow\quad \mathsf{N}\ f \cdot c = h\ , \tag{6.5}$$

for all $f\colon C \to B$ and homomorphisms $h : (C,c) \to (\mathsf{N}\ B, \delta\ B)$. Have we seen an arrow of the form $\mathsf{N}\ f \cdot c$ before?

6.2 Background: Bialgebras

A bialgebra combines an algebra and a coalgebra with a common carrier. Bialgebras come in many flavours; we need the variant that combines F-algebras and coalgebras for a comonad N. The two functors have to interact coherently, described by a distributive law.

Distributive laws A distributive law $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$ of an endofunctor F over a comonad N is a natural transformation satisfying the two coherence conditions:

$$\epsilon \circ \mathsf{F}\ \cdot\ \lambda\ =\ \mathsf{F} \circ \epsilon\ , \tag{6.6a}$$

$$\delta \circ \mathsf{F}\ \cdot\ \lambda\ =\ \mathsf{N} \circ \lambda\ \cdot\ \lambda \circ \mathsf{N}\ \cdot\ \mathsf{F} \circ \delta\ . \tag{6.6b}$$

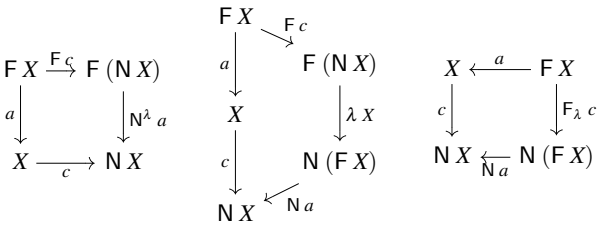
We can use λ to colift F to the category $\mathscr{C}_\mathsf{N}$. The coherence conditions guarantee that $\mathsf{F}_\lambda : \mathscr{C}_\mathsf{N} \to \mathscr{C}_\mathsf{N}$ preserves respect for ϵ and δ. Dually, λ induces the lifting $\mathsf{N}^\lambda : \mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C}) \to \mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C})$. Now, the coherence conditions ensure that N^λ is a comonad with $\bar{\epsilon}\ (a,A) = \epsilon\ A$ and $\bar{\delta}\ (a,A) = \delta\ A$. In particular, the lifted transformations $\bar{\epsilon} : \mathsf{N}^\lambda \dot{\to} \mathsf{Id}$ and $\bar{\delta} : \mathsf{N}^\lambda \dot{\to} \mathsf{N}^\lambda \circ \mathsf{N}^\lambda$ are F-algebra homomorphisms.

Bialgebras Let $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$ be a distributive law for the endofunctor F over the comonad N. A *λ-bialgebra* (a,X,c) consists of an F-algebra (a,X) and a coalgebra (X,c) for the comonad N such that the *pentagonal law* holds:

$$c \cdot a = \mathsf{N}\ a \cdot \lambda\ X \cdot \mathsf{F}\ c\ . \tag{6.7}$$

Loosely speaking, the law allows us to swap the algebra a and the coalgebra c. A λ-bialgebra homomorphism is both an F-algebra and an N-coalgebra homomorphism. λ-bialgebras and their homomorphisms form a category, denoted $\lambda\text{-}\mathbf{Bialg}(\mathscr{C})$.

The pentagonal law (6.7) also has two asymmetric renderings, which relate it to liftings and coliftings.



(6.8)

The diagram on the left shows that $c : (a,X) \to \mathsf{N}^\lambda\ (a,X)$ is an F-algebra homomorphism. Dually, the diagram on the right identifies $a : \mathsf{F}_\lambda\ (X,c) \to (X,c)$ as an N-coalgebra homomorphism. Thus, we can interpret the bialgebra (a,X,c) both as an algebra over a coalgebra $(a,(X,c))$, or as a coalgebra over an algebra $((a,X),c)$. Formally, we have the following isomorphisms of categories:

$$\mathsf{F}_\lambda\text{-}\mathbf{Alg}(\mathscr{C}_\mathsf{N}) \cong \lambda\text{-}\mathbf{Bialg}(\mathscr{C}) \cong (\mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C}))_{\mathsf{N}^\lambda}\ . \tag{6.9}$$

The alternative interpretations are useful to determine initial and final objects in $\lambda\text{-}\mathbf{Bialg}(\mathscr{C})$. To determine the initial object, we use the 'coalgebra over algebra' view, as categories of G-coalgebras have a trivial initial object: $(0, 0 \dashrightarrow \mathsf{G}\ 0)$, where 0 is the initial object in the underlying category and $0 \dashrightarrow \mathsf{G}\ 0$ the unique arrow from it. Consequently, $(in, \mu\mathsf{F}, fan)$ with $fan = (\!|\mathsf{N}^\lambda\ in|\!)$ is indeed initial.

6.3 Recursion Schemes from Comonads are Adjoint Folds

We now return to the proof of Theorem 6.2 using our new vocabulary to derive the unique solution. Somewhat surprisingly, as an immediate consequence of this proof, it turns out that rsfcs are an instance of adjoint folds, when previously, the two frameworks were thought of as being orthogonal [11]. Of course, the derivation is not strictly necessary, but it helps to relate the present development to prior work [30], and it hopefully helps to understand why rsfcs are an instance of adjoint folds. The development will follow the pattern we have already established.

First, we abstract away from the initial object $(in, \mu\mathsf{F}, fan)$, generalising to an arbitrary λ-bialgebra (a,A,c). The goal is to establish a bijection between arrows $f\colon A \to B$ satisfying $f \cdot a = b \cdot \mathsf{F}\ (\mathsf{N}\ f \cdot c)$ and λ-bialgebra homomorphisms $h : (a,A,c) \to (b_\sharp, \mathsf{N}\ B, \delta\ B)$, where $b_\sharp$ is a to-be-determined F-algebra. Now, we already know that arrows of type $f : A \to B$ and N-coalgebra homomorphisms $h : (A,c) \to (\mathsf{N}\ B, \delta\ B)$ are in one-to-one correspondence (6.5). So we identify $\mathsf{N}\ f \cdot c$ as the transpose of f and simplify f's equation to $f \cdot a = b \cdot \mathsf{F}\ h$. It remains to show that h is an F-algebra homomorphism of type $(a,A) \to (b_\sharp, \mathsf{N}\ B)$.

$$\begin{array}{ccc} \mathsf{F}\ A & \xrightarrow{\mathsf{F}\ h} & \mathsf{F}\ (\mathsf{N}\ B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ A & \xrightarrow[f]{} & B \end{array} \quad\Longleftrightarrow\quad \begin{array}{ccc} \mathsf{F}\ A & \xrightarrow{\mathsf{F}\ h} & \mathsf{F}\ (\mathsf{N}\ B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b_\sharp} \\ A & \xrightarrow[h]{} & \mathsf{N}\ B \end{array} \tag{6.10}$$

The strategy for the proof is clear: we have to transmogrify f into $\mathsf{N}\ f \cdot c$. Thus, we apply N to both sides of $f \cdot a = b \cdot \mathsf{F}\ h$ and then 'swap' a and c using the pentagonal law (6.7).

$$\begin{array}{ll} & f \cdot a = b \cdot \mathsf{F}\ h \\ \Longrightarrow & \quad \{\ \mathsf{N}\ \text{functor}\ \} \\ & \mathsf{N}\ f \cdot \mathsf{N}\ a = \mathsf{N}\ b \cdot \mathsf{N}\ (\mathsf{F}\ h) \\ \Longrightarrow & \quad \{\ \text{Leibniz}\ \} \\ & \mathsf{N}\ f \cdot \mathsf{N}\ a \cdot \mathsf{F}_\lambda\ c = \mathsf{N}\ b \cdot \mathsf{N}\ (\mathsf{F}\ h) \cdot \mathsf{F}_\lambda\ c \\ \Longleftrightarrow & \quad \{\ a : \mathsf{F}_\lambda\ (A,c) \to (A,c)\ (6.7)\ \} \\ & \mathsf{N}\ f \cdot c \cdot a = \mathsf{N}\ b \cdot \mathsf{N}\ (\mathsf{F}\ h) \cdot \mathsf{F}_\lambda\ c \\ \Longleftrightarrow & \quad \{\ \mathsf{F}_\lambda\ h : \mathsf{F}_\lambda\ (A,c) \to \mathsf{F}_\lambda\ (\mathsf{N}\ B, \delta\ B) \text{ and } \mathsf{F}_\lambda\ h = \mathsf{F}\ h\ \} \\ & h \cdot a = \mathsf{N}\ b \cdot \mathsf{F}_\lambda\ (\delta\ B) \cdot \mathsf{F}\ h \end{array}$$

The proof makes essential use of the fact that a and h are N-coalgebra homomorphisms, and that F_λ preserves coalgebra homomorphisms. Along the way, we have derived a formula for $b_\sharp$:

$$b_\sharp = \mathsf{N}\ b \cdot \mathsf{F}_\lambda\ (\delta\ B) = \mathsf{N}\ b \cdot \lambda\ (\mathsf{N}\ B) \cdot \mathsf{F}\ (\delta\ B)\ . \tag{6.11}$$

We have to show that $(b_\sharp, \mathsf{N}\ B, \delta\ B)$ is a λ-bialgebra. Since $\mathsf{F}_\lambda\ (\mathsf{N}\ B, \delta\ B)$ is a coalgebra for the comonad N, we can conclude using (6.5) that $b_\sharp$ is a coalgebra homomorphism of type $\mathsf{F}_\lambda\ (\mathsf{N}\ B, \delta\ B) \to (\mathsf{N}\ B, \delta\ B)$, which establishes the desired result. Furthermore $b = \epsilon\ B \cdot b_\sharp$, which allows us to complete the proof.

$$\begin{array}{ll} & h \cdot a = b_\sharp \cdot \mathsf{F}\ h \\ \Longrightarrow & \quad \{\ \text{Leibniz}\ \} \\ & \epsilon\ B \cdot h \cdot a = \epsilon\ B \cdot b_\sharp \cdot \mathsf{F}\ h \\ \Longleftrightarrow & \quad \{\ f = \epsilon\ B \cdot h \text{ and } b = \epsilon\ B \cdot b_\sharp\ \} \\ & f \cdot a = b \cdot \mathsf{F}\ h \end{array}$$

We have discovered an important fact: b and $b_\sharp$ are also related by the Eilenberg-Moore adjunction $\mathsf{U}_\mathsf{N} \dashv \mathsf{Cofree}_\mathsf{N}$! Using the notation for adjuncts, the right-hand side of (6.10) reads $\lfloor f \rfloor \cdot a = \lfloor b \rfloor \cdot \mathsf{F}\ \lfloor f \rfloor$. This looks suspiciously like the right-hand side of (5.7), which relates trahos (adjoint folds) and homomorphisms. However, the original equation for f does not seem to fit into the picture. This

is because it omits the forgetful functor $\mathsf{U_N}$. If we make it explicit, we obtain the the following bijection, which is indeed an instance of (5.7).

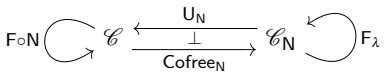
$$\begin{array}{ccc} \mathsf{U_N}\,(\mathsf{F}_\lambda\,(A,c)) & \xrightarrow{\mathsf{U_N}\,(\mathsf{F}_\lambda\,\lfloor f\rfloor)} & \mathsf{U_N}\,(\mathsf{F}_\lambda\,(\mathsf{Cofree_N}\,B)) \\ {\scriptstyle \mathsf{U_N}\,a}\downarrow & & \downarrow{\scriptstyle b} \\ \mathsf{U_N}\,(A,c) & \xrightarrow[f]{} & B \end{array}$$

$$\iff \quad \begin{array}{ccc} \mathsf{F}_\lambda\,(A,c) & \xrightarrow{\mathsf{F}_\lambda\,\lfloor f\rfloor} & \mathsf{F}_\lambda\,(\mathsf{Cofree_N}\,B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle \lfloor b\rfloor} \\ (A,c) & \xrightarrow[\lfloor f\rfloor]{} & \mathsf{Cofree_N}\,B \end{array}$$

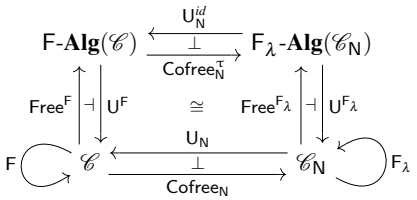
If we simplify the composition of functors using $\mathsf{U_N}\circ\mathsf{F}_\lambda = \mathsf{F}\circ\mathsf{U_N}$ and $\mathsf{U_N}\circ\mathsf{F}_\lambda\circ\mathsf{Cofree_N} = \mathsf{F}\circ\mathsf{U_N}\circ\mathsf{Cofree_N} = \mathsf{F}\circ\mathsf{N}$, we obtain the original equivalence (6.10). The somewhat pedantic diagrams above explicate all the information implicit in (6.10). For example, we can read off that a is an N-coalgebra homomorphism.

The *second* step should be routine by now. If we instantiate (a,A,c) to the initial λ-bialgebra $(in,\mu\mathsf{F},fan)$, we obtain that the unique solution of the original equation (6.1) is $f = \lceil\!(\lfloor b\rfloor)\!\rceil$ or, expressed using the vocabulary of [30], $f = \epsilon\,B\cdot(\!(b_\sharp)\!)$. We record

Theorem 6.3. *A recursion scheme from the comonad* N *and the distributive law* $\lambda : \mathsf{F}\circ\mathsf{N}\dot{\to}\mathsf{N}\circ\mathsf{F}$ *can be framed as an adjoint fold based on the Eilenberg-Moore adjunction* $\mathsf{U_N}\dashv\mathsf{Cofree_N}$*, using the canonical control functor* $\mathsf{U_N}\circ\mathsf{F}_\lambda\circ\mathsf{Cofree_N} = \mathsf{F}\circ\mathsf{N}$.



Let us conclude the section by investigating an alternative control functor: since $\mathsf{U_N}\circ\mathsf{F}_\lambda = \mathsf{F}\circ\mathsf{U_N}$, the functor F itself can be used as the control! For this case the distributive law σ is an isomorphism, even an identity, so we can invoke the machinery of Section 5.5 and lift the adjunction $\mathsf{U_N}\dashv\mathsf{Cofree_N}$ to an adjunction between categories of algebras. The conjugate of $\sigma = id$ is just λ, we have $\mathsf{U_N}\circ\tau = \lambda$. (The coherence condition (6.6b) shows that λ is an N-coalgebra homomorphism of type $\mathsf{F}_\lambda\circ\mathsf{Cofree_N}\dot{\to}\mathsf{Cofree_N}\circ\mathsf{F}$.)



In the upper right corner we find the category of λ-bialgebras (6.9). This shows that the underlying functor λ-**Bialg**$(\mathscr{C}) \to \mathsf{F}$-**Alg**$(\mathscr{C})$, which forgets about the algebra part, has a right adjoint. Since right adjoints preserve final objects, we immediately obtain that $\mathsf{Cofree_N^\tau}\,(\mathsf{F}\,1\dashrightarrow 1,1)$ is the final bialgebra.

7. When is an Adjoint Fold an Rsfc?

We have seen that all rsfcs are adjoint folds; and indeed, previous work has shown that the two share some connection—zygomorphisms have been modelled both by using rsfcs [30] and as adjoint folds [11]. But what about the reverse direction: when is an adjoint fold also modelled by using an rsfc? In this section we show that an adjoint fold based on the canonical control functor can be captured as an rsfc, if additionally a distributive isomorphism exists.

An adjoint fold is based on an adjunction $\mathsf{L}\dashv\mathsf{R}$, and an rsfc on a comonad. Thus, using Huber's result, an obvious choice for the comonad is $\mathsf{N} = \mathsf{L}\circ\mathsf{R}$. However, we also need to manufacture a distributive law $\lambda : \mathsf{F}\circ\mathsf{N}\dot{\to}\mathsf{N}\circ\mathsf{F}$. Now, one can show that a conjugate pair $\sigma\dashv\tau$ of distributive laws, *where* σ *is an isomorphism*, induces a distributive law for an endofunctor over comonad [32]. Consequently, we assume the following data (we rename F to C to bring the subsequent development in line with Section 5.5):

$$\sigma : \mathsf{L}\circ\mathsf{D}\cong\mathsf{C}\circ\mathsf{L} \;\dashv\; \tau : \mathsf{D}\circ\mathsf{R}\dot{\to}\mathsf{R}\circ\mathsf{C}\ .$$

These are the same assumptions as for the type fusion rule, Corollary 5.5. We shall see shortly that this is not a mere coincidence.

Under these assumptions we aim to show that the following two diagrams are equivalent. On the left we have the diagram for adjoint folds based on the canonical control functor (5.7); on the right we have the diagram for recursion schemes from comonads (6.1).

$$\begin{array}{ccc} \mathsf{L}\,(\mathsf{D}\,(\mu\mathsf{D})) & \xrightarrow{\mathsf{L}\,(\mathsf{D}\,\lfloor x\rfloor)} & \mathsf{L}\,(\mathsf{D}\,(\mathsf{R}\,B)) \\ {\scriptstyle \mathsf{L}\,in}\downarrow & & \downarrow{\scriptstyle b} \\ \mathsf{L}\,(\mu\mathsf{D}) & \xrightarrow[x]{} & B \end{array} \qquad \iff \qquad \begin{array}{ccc} \mathsf{C}\,(\mu\mathsf{C}) & \xrightarrow{\mathsf{C}\,(\mathsf{N}\,f\cdot fan)} & \mathsf{C}\,(\mathsf{N}\,B) \\ {\scriptstyle in}\downarrow & & \downarrow{\scriptstyle b\cdot\sigma^\circ\,(\mathsf{R}\,B)\quad b'} \\ \mu\mathsf{C} & \xrightarrow[f]{} & B \end{array}$$

The functions x and f are related by the isomorphism $\mathsf{L}\,(\mu\mathsf{D})\cong\mu\mathsf{C}$, provided by Corollary 5.5. The main task is to relate the upper arrows. To this end we derive a simple formula for *fan*, which uses the assumption that $\mathsf{N} = \mathsf{L}\circ\mathsf{R}$ is a composition of adjoint functors.

But first, we have to set up the infrastructure. From the data above we can generate two distributive laws [32]:

$$\alpha \;=\; \tau - \sigma^\circ \;=\; \mathsf{R}\circ\sigma^\circ\cdot\tau\circ\mathsf{L} \;:\; \mathsf{D}\circ\mathsf{M}\dot{\to}\mathsf{M}\circ\mathsf{D}\ , \tag{7.1a}$$

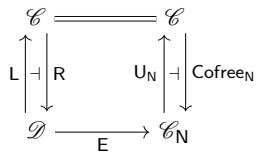
$$\gamma \;=\; \sigma^\circ - \tau \;=\; \mathsf{L}\circ\tau\cdot\sigma^\circ\circ\mathsf{R} \;:\; \mathsf{C}\circ\mathsf{N}\dot{\to}\mathsf{N}\circ\mathsf{C}\ . \tag{7.1b}$$

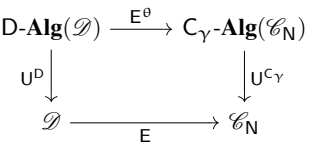
The distributive law γ satisfies the two requirements for λ (6.6) (note that $\mathsf{N}^\gamma = \mathsf{L}^{\sigma^\circ}\circ\mathsf{R}^\tau$); the distributive law α of an endofunctor over a monad enjoys analogous conditions:

$$\alpha\cdot\mathsf{D}\circ\eta \;=\; \eta\circ\mathsf{D}\ , \tag{7.2a}$$

$$\alpha\cdot\mathsf{D}\circ\mu \;=\; \mu\circ\mathsf{D}\cdot\mathsf{M}\circ\alpha\cdot\alpha\circ\mathsf{M}\ . \tag{7.2b}$$

Next we construct an initial γ-bialgebra. Since $\mathsf{L}\,(\mu\mathsf{D})\cong\mu\mathsf{C}$, we know that $(\mathsf{L}^{\sigma^\circ}\,in,\mathsf{L}\,(\mu\mathsf{D}))$ is initial in C-**Alg**$(\mathscr{C})$. To determine a formula for *fan*, the corresponding coalgebra part, we have to delve a bit deeper into the theory. The Eilenberg-Moore adjunction $\mathsf{U_N}\dashv\mathsf{Cofree_N}$ has an important property: it is the largest adjunction that generates N, in the sense that for every adjunction $\mathsf{L}\dashv\mathsf{R}$ there is a unique adjoint square from $\mathsf{L}\dashv\mathsf{R}$ to $\mathsf{U_N}\dashv\mathsf{Cofree_N}$, see the diagram on the left below. (Since the distributive laws of the adjoint square are identities, it is actually a so-called *map of adjunctions*.)





The so-called *comparison functor* $\mathsf{E} : \mathscr{D}\to\mathscr{C}_\mathsf{N}$ is defined

$$\mathsf{E}\,A = (\mathsf{L}\,A,\mathsf{L}\,(\eta\,A))\ , \tag{7.3a}$$

$$\mathsf{E}\,f = \mathsf{L}\,f\ . \tag{7.3b}$$

Since the distributive laws of the adjoint square are identities, we have $\mathsf{U_N}\circ\mathsf{E} = \mathsf{Id}\circ\mathsf{L}$ and $\mathsf{E}\circ\mathsf{R} = \mathsf{Cofree_N}\circ\mathsf{Id}$.

Note that the carrier of $\mathsf{E}\,A$ is $\mathsf{L}\,A$, which suggests that the coalgebra part of the initial γ-bialgebra is perhaps just $\mathsf{L}\,(\eta\,(\mu\mathsf{D}))$.

Then it remains to verify that $\mathsf{L}^{\sigma^\circ}\ in$ and $\mathsf{L}\ (\eta\ (\mu\mathsf{D}))$ satisfy the pentagonal law. We prove, in fact, a slightly more general result: we show that $(\mathsf{L}^{\sigma^\circ}\ a, \mathsf{L}\ A, \mathsf{L}\ (\eta\ A))$ is a γ-bialgebra, by lifting the comparison functor E to categories of algebras, see diagram on the right above. To this end, we need a distributive law $\theta : \mathsf{E}\circ\mathsf{D} \leftarrow \mathsf{C}_\gamma \circ \mathsf{E}$. We claim that σ° itself fits the bill: $\mathsf{U}_\mathsf{N}\circ\theta = \sigma^\circ$. (Note that $\mathsf{U}_\mathsf{N}\circ\mathsf{E}\circ\mathsf{D} \leftarrow \mathsf{U}_\mathsf{N}\circ\mathsf{C}_\gamma\circ\mathsf{E} = \mathsf{L}\circ\mathsf{D}\leftarrow\mathsf{C}\circ\mathsf{L}$). In other words, we have to show that σ° is a natural N-coalgebra homomorphism of type $\mathsf{E}\circ\mathsf{D} \leftarrow \mathsf{C}_\gamma\circ\mathsf{E}$. Plugging in the definitions, we reason

$$\begin{array}{ll} & \mathsf{L}\circ\eta\circ\mathsf{D}\ \cdot\ \sigma^\circ \\ = & \quad \{\ \alpha \text{ respects } \eta\ (7.2a)\ \} \\ & \mathsf{L}\circ\alpha\ \cdot\ \mathsf{L}\circ\mathsf{D}\circ\eta\ \cdot\ \sigma^\circ \\ = & \quad \{\ \sigma^\circ \text{ is natural}\ \} \\ & \mathsf{L}\circ\alpha\ \cdot\ \sigma^\circ\circ\mathsf{M}\ \cdot\ \mathsf{C}\circ\mathsf{L}\circ\eta \\ = & \quad \{\ \sigma^\circ - \alpha = \sigma^\circ - \tau - \sigma^\circ = \gamma - \sigma^\circ\ \} \\ & \mathsf{N}\circ\sigma^\circ\ \cdot\ \gamma\circ\mathsf{L}\ \cdot\ \mathsf{C}\circ\mathsf{L}\circ\eta\ . \end{array}$$

We have derived the following attractive definition of *fan*:

$$fan = \mathsf{L}\ (\eta\ (\mu\mathsf{D}))\ , \tag{7.4}$$

which is in most cases a much more efficient implementation than $(\!|\mathsf{N}\ in\cdot\lambda\ (\mu\mathsf{F})|\!)$.

We are now ready to show that adjoint folds are recursion schemes from comonads, provided that a distributive isomorphism $\sigma : \mathsf{L}\circ\mathsf{D} \cong \mathsf{C}\circ\mathsf{L}$ exists. Here is the diagram for rsfcs with applications of the isomorphism σ made explicit.

$$\begin{array}{ccccccc} \mathsf{L}\ (\mathsf{D}\ (\mu\mathsf{D})) & \xrightarrow{\sigma\ (\mu\mathsf{D})} & \mathsf{C}\ (\mathsf{L}\ (\mu\mathsf{D})) & \xrightarrow{\mathsf{C}\ (\mathsf{N}\ x\cdot fan)} & \mathsf{C}\ (\mathsf{N}\ B) & \xrightarrow{\sigma^\circ\ (\mathsf{R}\ B)} & \mathsf{L}\ (\mathsf{D}\ (\mathsf{R}\ B)) \\ \downarrow{\scriptstyle \mathsf{L}\ in} & & \downarrow{\scriptstyle \mathsf{L}^{\sigma^\circ}\ in} & & \downarrow{\scriptstyle b\cdot\sigma^\circ\ (\mathsf{R}\ B)} & & \downarrow{\scriptstyle b} \\ \mathsf{L}\ (\mu\mathsf{D}) & = & \mathsf{L}\ (\mu\mathsf{D}) & \xrightarrow[x]{} & B & = & B \end{array}$$

Thus, it remains to show that $\mathsf{L}\ \lfloor x \rfloor = \mathsf{N}\ x\cdot fan$.

$$\begin{array}{ll} & \mathsf{L}\ \lfloor x \rfloor \\ = & \quad \{\ \lfloor - \rfloor \text{ expressed in terms of } \eta;\ (3.4a) \text{ with } k,f,h := x, id, id\ \} \\ & \mathsf{L}\ (\mathsf{R}\ x\cdot\eta\ (\mu\mathsf{D})) \\ = & \quad \{\ \mathsf{L} \text{ functor and definition of } fan\ (7.4)\ \} \\ & \mathsf{N}\ x\cdot fan \end{array}$$

We record the result in the following

Theorem 7.1. *An adjoint fold based on the adjunction* $\mathsf{L}\dashv\mathsf{R}$ *and the canonical control functor can be framed as a recursion scheme for the comonad* $\mathsf{N} = \mathsf{L}\circ\mathsf{R}$ *if there is a control functor* C *and a conjugate pair of distributive laws with* σ *an isomorphism:*

$$\sigma : \mathsf{L}\circ\mathsf{D} \cong \mathsf{C}\circ\mathsf{L}\ \ \dashv\ \ \tau : \mathsf{D}\circ\mathsf{R} \dot{\to} \mathsf{R}\circ\mathsf{C}\ .$$

The distributive law $\lambda : \mathsf{C}\circ\mathsf{N} \dot{\to} \mathsf{N}\circ\mathsf{C}$ *is given by* $\sigma^\circ - \tau$.

Example 7.2. Mutumorphisms are an instance of rsfcs using $\sigma = id : \Delta\circ\mathsf{D} = \mathsf{D}^2\circ\Delta$, see Example 5.6. Since the isomorphism is an identity, we can transform the diagram for adjoints folds almost directly into a corresponding diagram for rsfcs ($A := \mu\mathsf{D}$).

$$\begin{array}{ccc} \Delta\ (\mathsf{D}\ A) & \xrightarrow{\Delta\ (\mathsf{D}\ \lfloor x \rfloor)} & \Delta\ (\mathsf{D}\ ((\times)\ B)) \\ \downarrow{\scriptstyle \Delta\ in} & & \downarrow{\scriptstyle b} \\ \Delta\ A & \xrightarrow[x]{} & B \end{array} \quad \Longleftrightarrow \quad \begin{array}{ccc} \mathsf{D}^2\ (\Delta\ A)) & \xrightarrow{\mathsf{D}^2\ (\Delta\ \lfloor x \rfloor)} & \mathsf{D}^2\ (\Delta\ ((\times)\ B)) \\ \downarrow{\scriptstyle \Delta\ in} & & \downarrow{\scriptstyle b} \\ \Delta\ A & \xrightarrow[x]{} & B \end{array}$$

In the upper right corner we discover the comonad $\mathsf{N} = \Delta\circ(\times)$, which works over a product category. As $fan = \Delta\ (id \vartriangle id)$, we have $\Delta\ \lfloor x \rfloor = \Delta\ ((\times)\ x\cdot(id\vartriangle id)) = \mathsf{N}\ x\cdot fan$. If we identify $\Delta\ (\mu\mathsf{D})$ and $\mu\mathsf{D}^2$ so that $\Delta\ in = in$, the diagram for rsfcs emerges. □

We have seen in the previous section that an rsfc (based on a comonad N and a distributive law $\lambda : \mathsf{F}\circ\mathsf{N} \dot{\to} \mathsf{N}\circ\mathsf{F}$) can be framed as an adjoint fold, which in turn is based on the Eilenberg-Moore adjunction $\mathsf{U}_\mathsf{N} \dashv \mathsf{Cofree}_\mathsf{N}$. Now, what happens if we go round in a circle, instantiating the development above to $\mathsf{D} = \mathsf{F}_\lambda$, $\mathsf{C} = \mathsf{F}$, and $id : \mathsf{U}_\mathsf{N}\circ\mathsf{F}_\lambda = \mathsf{F}\circ\mathsf{U}_\mathsf{N}$? Recall that $\mathsf{U}_\mathsf{N}\circ\tau = \lambda$; consequently,

$$\gamma\ =\ \sigma^\circ - \tau\ =\ \mathsf{U}_\mathsf{N}\circ\tau\ \cdot\ id^\circ\circ\mathsf{Cofree}_\mathsf{N}\ =\ \lambda\ .$$

Hence, we obtain back the original rsfc!

We have seen that mutumorphisms based on $\Delta\dashv(\times)$ can be modelled by using an rsfc. Of course, this does not work for every adjunction. As an example, consider the curry adjunction. One would need a control functor C such that:

$$\sigma : (-\times P)\circ\mathsf{D} \cong \mathsf{C}\circ(-\times P)\ .$$

Such a control functor is not guaranteed to exist for all datatypes.

8. Related Work

Adjoint folds Mendler-style adjoint folds first appeared in a paper by Bird and Paterson [4], where they were used to show that generalised folds are uniquely defined. (Somewhat ironically, adjoint folds were *only* used in the proofs, not as a general recursion principle.) The algebraic variant of adjoint folds that we have employed throughout was introduced by Matthes and Uustalu [20] under the name *generalised iteration*. The first author of the present paper explored the design space of adjoint folds [11], identifying the adjunctions underlying various recursion schemes. The paper also shows how to combine recursion schemes by combining the underlying adjunctions. Alas, we wrote "However, we cannot reasonably expect that adjoint (un)folds subsume all existing species of morphisms. For instance, a largely orthogonal extension of standard folds are *recursion schemes from comonads*."

Recursion schemes from comonads Recursion schemes from comonads are due to Uustalu et al. [30]. Simultaneously and independently, Bartels [2] introduced the dual construction under name *λ-coiteration*. Technically, his work is closest to ours in the use of λ-bialgebras, although our proofs differ considerably in that we make use of the Eilenberg-Moore construction. Bartels also discussed variations of the scheme that do not rely on a monad structure. These and further variations were used in a recent ICFP paper [12] to prove the unique fixed-point principle correct.

Categorical fixed-point calculus The roots of the initial algebra approach to the semantics of datatypes can be traced back to the work of Lambek [16] on fixed points in categories. Lambek suggests that lattice theory provides a fruitful source of inspiration for results in category theory. These ideas were picked up by Backhouse et al. [1], where a number of lattice-theoretic fixed-point rules were generalised to categories, type fusion being one of them.

Category theory Most of the category theory utilised in this paper is fairly standard—[17] is a good reference—except perhaps the material on distributive laws and conjugates. An extensive account of the relationship between adjunctions and monads is provided by Vidal and Tur [32]. Roughly speaking, they show that the Eilenberg-Moore construction is a right biadjoint to the Huber construction.

9. Conclusion

As one might expect, everything dualises elegantly. There is insufficient space to tell the full story here; but the essential fact is that adjoint unfolds subsume anamorphisms, apomorphisms, generalised unfolds, λ-coiteration, futumorphisms, and other dual patterns that have not yet been formally christened. Table 1 provides an overview of the various morphisms and their duals, and shows how they are captured in the framework of adjoint (un)folds.

(a)				(b)		
	catamorphism [10, 19]	$\mathsf{Id} \dashv \mathsf{Id}$	*depth*		anamorphism [10, 19]	$\mathsf{Id} \dashv \mathsf{Id}$
	mutumorphism [8]	$\Delta \dashv (\times)$	*even/odd*		(mutumorphism)	$(+) \dashv \Delta$
	special case: zygomorphism [18]		*perfect*		—	
	special case: paramorphism [21]		*wc*		apomorphism [31]	
	fold with a parameter [25]	$- \times P \dashv (-)^P$	*cat*, *depths*		—	
	generalised fold [4]	$(- \circ \mathsf{P}) \dashv \mathsf{Ran}_\mathsf{P}$	*total*		(generalised unfold)	$\mathsf{Lan}_\mathsf{P} \dashv (- \circ \mathsf{P})$
	recursion scheme from a comonad [30]	$\mathsf{U}_\mathsf{N} \dashv \mathsf{Cofree}_\mathsf{N}$			λ-coiteration [2]	$\mathsf{Free}^\mathsf{M} \dashv \mathsf{U}^\mathsf{M}$
	special case: histomorphism [26]		*knapsack*		*special case:* futumorphism [26]	

Table 1. Adjoint folds (a) and unfolds (b); Lan_P and Ran_P are left and right Kan extensions, $\mathsf{Free}^\mathsf{M} \dashv \mathsf{U}^\mathsf{M}$ is the Eilenberg-Moore adjunction

This paper shows again the importance of adjunctions. They have played a pivotal role in the categorical analysis of logic; we believe that will prove just as important in the theory of programming. The unification of recursion schemes we have presented is mathematically satisfying: in the economy of expression it provides (for example, for reasoning about products in F-$\mathbf{Alg}(\mathscr{C})$), and especially in the simple reassurance it provides through things just fitting together in the right way. But it is more than merely an intellectual curiosity: the additional structure we have uncovered promises concrete returns, too—for example, through general techniques for combining different recursion schemes, by composing the corresponding adjunctions. In practice, most functions do indeed use a combination of recursion schemes (in particular, functions over parametric datatypes).

In future work, we intend to explore the calculational properties of adjoint folds; our earlier paper [11] lists several laws, and it would be interesting to know whether the laws for recursion schemes from comonads of Uustalu et al. [30] are instances of these. Another interesting direction is to explore the use of *recursive* coalgebras [5] (or *corecursive* algebras) in λ-bialgebras.

Acknowledgments

This work has been funded by EPSRC grant number EP/J010995/1, on Unifying Theories of Generic Programming. The authors would like to thank the anonymous reviewers and José Pedro Magalhães for their helpful and constructive comments.

References

[1] R. Backhouse, M. Bijsterveld, R. van Geldrop, and J. van der Woude, "Categorical fixed point calculus," in *CTCS*, ser. LNCS, vol. 953. Springer, 1995, pp. 159–179.

[2] F. Bartels, "Generalised coinduction," *Mathematical Structures in Computer Science*, vol. 13, pp. 321–348, 2003.

[3] R. Bird and O. de Moor, *Algebra of Programming*. London: Prentice Hall, 1997.

[4] R. Bird and R. Paterson, "Generalised folds for nested datatypes," *Formal Aspects of Computing*, vol. 11, no. 2, pp. 200–222, 1999.

[5] V. Capretta, T. Uustalu, and V. Vene, "Recursive coalgebras from comonads," *Information and Computation*, vol. 204, no. 4, pp. 437–468, 2006.

[6] S. Eilenberg and J. C. Moore, "Adjoint functors and triples," *Illinois J. Math*, vol. 9, no. 3, pp. 381–398, 1965.

[7] M. M. Fokkinga, "Law and order in algorithmics," Ph.D. dissertation, University of Twente, Feb. 1992.

[8] ——, "Tupling and mutumorphisms," *The Squiggolist*, vol. 1, no. 4, pp. 81–82, Jun. 1990.

[9] J. Gibbons, "Generic downwards accumulations," *Science of Computer Programming*, vol. 37, no. 1-3, pp. 37–65, 2000.

[10] T. Hagino, "Category theoretic approach to data types," Ph.D. dissertation, University of Edinburgh, 1987.

[11] R. Hinze, "Adjoint folds and unfolds—an extended study," *Science of Computer Programming*, Aug. 2012.

[12] R. Hinze and D. W. James, "Proving the unique fixed-point principle correct: an adventure with category theory," in *ICFP*. ACM, 2011, pp. 359–371.

[13] P. J. Huber, "Homotopy theory in general categories," *Math. Ann.*, vol. 144, pp. 361–385, 1961.

[14] D. M. Kan, "Adjoint functors," *Trans. AMS*, vol. 87, no. 2, pp. 294–329, 1958.

[15] H. Kleisli, "Every standard construction is induced by a pair of adjoint functors," *Proc. AMS*, vol. 16, no. 3, pp. 544–546, Jun. 1965.

[16] J. Lambek, "A fixpoint theorem for complete categories," *Math. Zeitschr.*, vol. 103, pp. 151–161, 1968.

[17] S. Mac Lane, *Categories for the Working Mathematician*, 2nd ed., ser. Graduate Texts in Mathematics. Berlin: Springer-Verlag, 1998.

[18] G. Malcolm, "Algebraic data types and program transformation," Ph.D. dissertation, University of Groningen, 1990.

[19] ——, "Data structures and program transformation," *Science of Computer Programming*, vol. 14, no. 2–3, pp. 255–280, 1990.

[20] R. Matthes and T. Uustalu, "Substitution in non-wellfounded syntax with variable binding," *TCS*, vol. 327, no. 1–2, pp. 155–174, 2004.

[21] L. Meertens, "Paramorphisms," *Formal Aspects of Computing*, vol. 4, pp. 413–424, 1992.

[22] E. Meijer, M. Fokkinga, and R. Paterson, "Functional programming with bananas, lenses, envelopes and barbed wire," in *FPLCA*, ser. LNCS, vol. 523. Springer, 1991, pp. 124–144.

[23] N. P. Mendler, "Inductive types and type constraints in the second-order lambda calculus," *Ann. Pure Appl. Logic*, vol. 51, no. 1–2, pp. 159–172, 1991.

[24] P. H. Palmquist, "The double category of adjoint squares," in *Midwest Category Seminar V*, ser. LNM. Springer, 1971, vol. 195, pp. 123–153.

[25] A. Pardo, "Generic accumulations," in *Working Conference on Generic Programming*, vol. 243. Kluwer Academic Publishers, Jul. 2002, pp. 49–78.

[26] T. Uustalu and V. Vene, "Primitive (co)recursion and course-of-value (co)iteration, categorically," *Informatica, Lith. Acad. Sci.*, vol. 10, no. 1, pp. 5–26, 1999.

[27] ——, "Comonadic notions of computation," in *CMCS*, ser. ENTCS, vol. 203(5), 2008, pp. 263–284.

[28] ——, "Mendler-style inductive types, categorically," *Nordic J. Comput.*, vol. 6, pp. 343–361, 1999.

[29] ——, "The recursion scheme from the cofree recursive comonad," *ENTCS*, vol. 229, no. 5, pp. 135–157, 2011, proceedings of the Second Workshop on Mathematically Structured Functional Programming (MSFP 2008).

[30] T. Uustalu, V. Vene, and A. Pardo, "Recursion schemes from comonads," *Nordic J. Comput.*, vol. 8, pp. 366–390, Sep. 2001.

[31] V. Vene and T. Uustalu, "Functional programming with apomorphisms (corecursion)," *Proceedings of the Estonian Academy of Sciences: Physics, Mathematics*, vol. 47, no. 3, pp. 147–161, 1998.

[32] J. C. Vidal and J. S. Tur, "Kleisli and Eilenberg-Moore constructions as parts of biadjoint situations," *Extracta Mathematicae*, vol. 25, no. 1, pp. 1–61, 2010.