Under consideration for publication in J. Functional Programming

Unifying Structured Recursion Schemes

An Extended Study

RALF HINZE
Department of Computer Science, University of Oxford

NICOLAS WU
Department of Computer Science, University of Bristol
(*e-mail:* ralf.hinze@cs.ox.ac.uk, nicolas.wu@bristol.ac.uk)

Abstract

Folds and unfolds have been understood as fundamental building blocks for total programming, and have been extended to form an entire zoo of specialised structured recursion schemes. A great number of these schemes were unified by the introduction of adjoint folds, but more exotic beasts such as recursion schemes from comonads proved to be elusive. In this paper, we show how the two canonical derivations of adjunctions from (co)monads yield recursion schemes of significant computational importance: monadic catamorphisms come from the Kleisli construction, and more astonishingly, the elusive recursion schemes from comonads come from the Eilenberg-Moore construction. Thus we demonstrate that adjoint folds are more unifying than previously believed.

1 Introduction

Functional programmers have long realised that the full expressive power of recursion is untamable, and so intensive research has been carried out into the identification of an entire zoo of structured recursion schemes that are well-behaved and more amenable to program comprehension and analysis (Meijer *et al.*, 1991).

The foundational structured recursion operators are catamorphisms or folds and anamorphisms or unfolds (Hagino, 1987; Malcolm, 1990b): they make termination or progress manifest, and enjoy many useful calculational properties which would otherwise have to be established afresh for each new application.

However, catamorphisms and anamorphisms are relatively restricted. There are many other structured patterns of recursion, equally well behaved and worth capturing, that do not quite fit the scheme. Variations on catamorphisms that have been proposed in the past include folds with parameters and accumulating folds (Pardo, 2002), which may depend on constant or varying additional arguments; mutumorphisms (Fokkinga, 1990), which are pairs of mutually recursive functions; zygomorphisms (Malcolm, 1990a), which consist of a main recursive function and an auxiliary one on which it depends; paramorphisms (Meertens, 1992), in which the body of structural recursion has access to immediate subterms as well as to their images under the recursion; histomorphisms (Uustalu & Vene, 1999b), in which the body has access to the recursive images of all subterms, not just the immediate ones;

and so-called generalised folds (Bird & Paterson, 1999), which use polymorphic recursion to handle nested datatypes.

As variations on anamorphisms, there are apomorphisms (Vene & Uustalu, 1998), which may generate subterms monolithically rather than step by step; futumorphisms (Uustalu & Vene, 1999b), which may generate multiple levels of a subterm in a single step, rather than just one; and many other anonymous schemes that dualize better known inductive patterns of recursion.

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 (Uustalu *et al.*, 2001) which we call 'rsfcs' for short. Comonads capture the general idea of 'evaluation in context' (Uustalu & Vene, 2008), and rsfcs make contextual information available to the body of the recursion. This pattern subsumes zygomorphisms and histomorphisms.

A more recent attempt (Hinze, 2013) 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. Dually, adjoint unfolds involve a right adjoint, and capture the production of a data structure in context; they subsume all the above variations on anamorphisms.

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.

This article is an extended and revised version of (Hinze *et al.*, 2013), which in turn draws on material from (Hinze, 2013), although the technical presentation is quite different, making essential use of liftings, distributive laws, and conjugates. Our technical contributions are as follows, where the new material in this extended study is indicated by an open bullet point:

- 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 a data structure to a control structure.
- ○ We show that monadic catamorphisms (Fokkinga, 1994) are an instance of adjoint folds using the Kleisli adjunction.
- We show that rsfcs (Uustalu *et al.*, 2001) are an instance of adjoint folds using the (co)Eilenberg-Moore adjunction.
- We state precisely the relationship to the (type) fusion rule of categorical fixed-point calculus (Backhouse *et al.*, 1995). 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 σ is a distributive isomorphism.
- ○ We explore the calculational properties of adjoint folds, and show how the well-established properties of other structured recursion schemes are instances of this more general theory.

- We demonstrate the dual notion of adjoint unfolds, where a distributive law $\tau : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C}$ connects a decision structure to a codata structure.

We give three different definitions of adjoint folds: one in terms of Mendler-style folds, and one in terms of conjugates which leads to a canonical definition. The conjugate-based definition makes the proofs of uniqueness of schemes easier, where 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, folds and unfolds, so no new theory is needed. 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). In addition, the unification also brings together the different calculational properties of recursion schemes under one umbrella, thus vastly reducing the number of laws required for calculation.

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 discusses mutumorphisms as a basic example of our unifying theory of adjoint folds, which is set out in Section 5; Section 6 relates the discussion to zygomorphisms. Section 7 shows that all rsfcs are adjoint folds, and Section 8 identifies those adjoint folds that are rsfcs; we explore the Kleisli adjunction and its relationship to monadic catamorphisms in Section 9; Section 10 discusses the calculational properties of adjoint folds; Section 11 shows the construction of adjoint unfolds; and finally, Section 12 discusses related work, and Section 13 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 widely appreciated *lingua franca* for codifying our categorical constructions as programs. Although Haskell conflates inductive and coinductive types, our categorical development will be careful to distinguish between the two.

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

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

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

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

Here, the second component of the input pair is the parameter; *cat* is not a fold because the pair argument is not of 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 (Gibbons, 2000); for example, replacing every element with a label of its depth (if the accumulator is initialised to 0):

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

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:

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

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 that is defined independently.

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

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:

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

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 involve 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 $\mathit{Perfect}\ \mathbb{N}$.

Anamorphism The dual of a catamorphism is an *anamorphism*, or *unfold*. This corecursion scheme builds a structure from a single seed, one level at a time.

$$
\begin{aligned}
&\mathit{from} :: \mathit{Int} \to [\mathit{Int}]\\
&\mathit{from}\ x = x : \mathit{from}\ (x+1)
\end{aligned}
$$

This example builds an infinite stream of increasing integers starting from a given number.

Apomorphism An *apomorphism* can immediately return a subterm in its result, rather than having to generate each part of the output structure through corecursion. An example is an insertion into a sorted tree structure, where elements in left branches are less than or equal to the value in a node, and values in right branches are greater.

$$
\begin{aligned}
&\mathit{insert} :: \mathbb{N} \to \mathit{Tree} \to \mathit{Tree}\\
&\mathit{insert}\ a\ \mathit{Empty} = \mathit{Node}\ \mathit{Empty}\ a\ \mathit{Empty}\\
&\mathit{insert}\ a\ (\mathit{Node}\ l\ b\ r) \mid a \leqslant b \quad\ = \mathit{Node}\ (\mathit{insert}\ a\ l)\ b\ r\\
&\qquad\qquad\qquad\qquad\ \mid \mathit{otherwise} = \mathit{Node}\ l\ b\ (\mathit{insert}\ a\ r)
\end{aligned}
$$

This behaves like the unfolding of a tree in one branch, growing a new tree from the element to be inserted and an existing tree, but in the other branch it immediately grafts a result.

Histomorphism Histomorphisms capture tabulation, as used in dynamic programming. For example, consider the unbounded knapsack problem: a knapsack of some fixed capacity is to be filled with items of varying weight and value. The goal is to maximise the total value of the items contained in the knapsack. Suppose there is an infinite supply of items whose weight and value can be drawn from the following list: $[(12,4),(1,2),(2,2),(1,1),(4,10)]$. A knapsack with a capacity of 15 can be filled to a maximum value of 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):

$$\begin{aligned}&\mathit{knapsack} :: [(\mathbb{N}, \mathbb{R})] \to \mathbb{N} \to \mathbb{R}\\&\mathit{knapsack}\ \mathit{wvs}\ c\\&\quad = \mathit{maximum}_0\ [v + \mathit{knapsack}\ \mathit{wvs}\ (c-w) \mid (w,v) \leftarrow \mathit{wvs}, w \leqslant c]\ .\end{aligned}$$

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

$$\begin{aligned}&\mathit{knapsack}\ \mathit{wvs}\ c = \mathit{table}\ !!\ c\ \textbf{where}\\&\quad \mathit{table} = [\mathit{ks}\ i \mid i \leftarrow [0\,..\,c]]\\&\quad \mathit{ks}\ i\ \ = \mathit{maximum}_0\ [v + \mathit{table}\ !!\ (i-w) \mid (w,v) \leftarrow \mathit{wvs}, w \leqslant i]\ .\end{aligned}$$

Lazy evaluation works out the data dependencies automatically. However, 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.

Monadic catamorphism Monadic catamorphisms allow a monadic computation to be threaded through the catamorphic traversal of a recursive structure. For example, the function *accumulate* takes a list of monadic actions and executes them each in turn, and returns a list of their results in the monadic context.

$$\begin{aligned}&\mathit{accumulate} :: \mathit{Monad}\ m \Rightarrow [m\ a] \to m\ [a]\\&\mathit{accumulate}\ [\,] \qquad\quad = \mathit{return}\ [\,]\\&\mathit{accumulate}\ (\mathit{mx} : \mathit{mxs}) = \mathit{mx} \gg\!= \lambda x \to \mathit{accumulate}\ \mathit{mxs} \gg\!= \lambda \mathit{xs} \to \mathit{return}\ (x : \mathit{xs})\end{aligned}$$

When the input list is empty, we simply return the empty list in a monadic context. Otherwise, we run the monadic computation at the head of the list, and return the result of appending it to the accumulation of executed values in the tail.

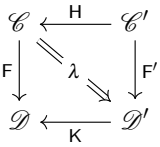
Now, the general question is whether the recursion equations above have unique solutions? The answer is yes for all of them. However, up to now the proofs in the literature have involved two seemingly incompatible techniques: most of the examples can be identified as adjoint folds; some of them (in particular *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 at least a basic knowledge of category theory, in that the reader should be familiar with the notions of functors, natural transformations, as well as 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.

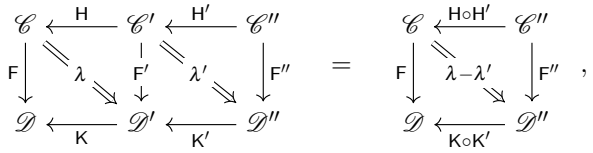
3.1 Functor Squares and Distributive Laws

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):



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

For brevity, we call λ a *distributive law*, even though the name is traditionally used for the special case in which opposite functors are monads (Beck, 1969) or also comonads (Turi & Plotkin, 1997), and which are subject to additional coherence conditions. 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 (we agree that $\circ$ binds tighter than $\cdot$):

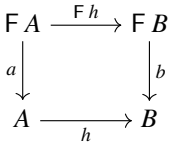
$$\lambda - \lambda' \ = \ \mathsf{K} \circ \lambda' \ \cdot \ \lambda \circ \mathsf{H}' \ .$$

This composition is associative, with $id_{\mathsf{F}} : \mathsf{F} \circ \mathsf{Id} \dot{\to} \mathsf{Id} \circ \mathsf{F}$ 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 $\mathsf{F} : \mathscr{C} \to \mathscr{C}$, an F-algebra is a pair (a,A), where $a : \mathsf{F}\,A \to A$ 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 $h : A \to B : \mathscr{C}$ such that $h \cdot a = b \cdot \mathsf{F}\,h$. Note, we sometimes annotate the category the arrows belong to by adding the category to the end of the signature.



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*. Initiality implies

that to each F-algebra, (a,A), there exists a unique F-homomorphism, $(\!|a|\!): (in, \mu\mathsf{F}) \to (a,A)$, 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 μTree, 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,

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

amounts to a simple renaming of constructors.

Dually, given an endofunctor $\mathsf{G} : \mathscr{C} \to \mathscr{C}$, a G-coalgebra is a pair (C,c), where $C : \mathscr{C}$ is the carrier and $c : C \to \mathsf{G}\ C$ is the action of the coalgebra. A G-homomorphism between coalgebras (C,c) and (D,d) is an arrow $h : C \to D : \mathscr{C}$ 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 unique morphism to each other G-algebra (C,c), called an *unfold*, is written $[\![c]\!]: (C,c) \to (\nu\mathsf{G}, \mathit{out})$.

The category F-**Alg**$(\mathscr{C})$ has more structure than $\mathscr{C}$. The forgetful or underlying functor $\mathsf{U}^{\mathsf{F}} : \mathsf{F}$-**Alg**$(\mathscr{C}) \to \mathscr{C}$ 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**$(\mathscr{C}) \to \mathscr{C}$.

Liftings and coliftings A functor $\mathsf{H} : \mathsf{F}$-**Alg**$(\mathscr{C}) \to \mathsf{G}$-**Alg**$(\mathscr{D})$ is called a *lifting* of the functor $\mathsf{H} : \mathscr{C} \to \mathscr{D}$ iff $\mathsf{H} \circ \mathsf{U}^{\mathsf{F}} = \mathsf{U}^{\mathsf{G}} \circ \mathsf{H}$. Given a distributive law $\lambda : \mathsf{H} \circ \mathsf{F} \leftarrow \mathsf{G} \circ \mathsf{H}$, 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 $\lambda = \alpha : \mathsf{F} \leftarrow \mathsf{G}$, are often written as α-**Alg**$(\mathscr{C}) : \mathsf{F}$-**Alg**$(\mathscr{C}) \to \mathsf{G}$-**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**$(\mathscr{C}) \to \mathsf{G}$-**Coalg**$(\mathscr{D})$ is a colifting of $\mathsf{H} : \mathscr{C} \to \mathscr{D}$ iff $\mathsf{U}_{\mathsf{G}} \circ \underline{\mathsf{H}} = \mathsf{H} \circ \mathsf{U}_{\mathsf{F}}$. Given $\lambda : \mathsf{H} \circ \mathsf{F} \dot{\to} \mathsf{G} \circ \mathsf{H}$ 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 (1958) and are so pervasive in the study of category theory that Mac Lane (1998, 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

$$\mathscr{C} \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{D} \quad ,$$

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:

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

for all $f : \mathsf{L}\, A \rightarrow B : \mathscr{C}$ and $g : A \rightarrow \mathsf{R}\, B : \mathscr{D}$. 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* $\epsilon : \mathsf{L} \circ \mathsf{R} \dot{\rightarrow} \mathsf{Id}$ and the *unit* $\eta : \mathsf{Id} \dot{\rightarrow} \mathsf{R} \circ \mathsf{L}$ of the adjunction. The units must satisfy the so-called triangle identities:

$$(\epsilon \circ \mathsf{L}) \cdot (\mathsf{L} \circ \eta) = \mathsf{L} \ , \tag{3.5a}$$

$$(\mathsf{R} \circ \epsilon) \cdot (\eta \circ \mathsf{R}) = \mathsf{R} \ . \tag{3.5b}$$

The equivalence (3.3) can also be framed in terms of the units:

$$f = \epsilon\, B \cdot \mathsf{L}\, g \iff \mathsf{R} f \cdot \eta\, A = g \ . \tag{3.6}$$

We explicitly instantiate a natural transformation to its component morphism by supplying the relevant object as a parameter, rather than as a subscript. Hence, $\epsilon\, B$ is the component of ϵ at the object B.

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

$$\mathsf{L}_2 \cong \mathsf{L}_1 \iff \mathsf{R}_1 \cong \mathsf{R}_2 \ . \tag{3.7}$$

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, and dually, right adjoints preserve final objects:

$$\mathsf{L}\,0 \cong 0 \ , \tag{3.8a}$$

$$\mathsf{R}\,1 \cong 1 \ . \tag{3.8b}$$

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 $\epsilon = (\mathit{outl}, \mathit{outr})$ arises from the projections. The split combinator should not be confused with diagonal functor, which is also denoted by a triangle.

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* F-*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, the *cofree* G-*coalgebra* arises as the right adjoint of U_{G}.

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

These adjunctions correspond to the following bijections:

$$\mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C})(\mathsf{U}^{\mathsf{F}}\,A, B) \cong \mathscr{C}(A, \mathsf{Free}^{\mathsf{F}}\,B) \ , \tag{3.9a}$$

$$\mathscr{C}(\mathsf{Cofree}_{\mathsf{G}}\,A, B) \cong \mathsf{G}\text{-}\mathbf{Coalg}(\mathscr{C})(A, \mathsf{U}_{\mathsf{G}}\,B) \ . \tag{3.9b}$$

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): $(\mathit{in}, \mu\mathsf{F}) \cong \mathsf{Free}^{\mathsf{F}}\,0$ (closed terms as open terms where the variables are drawn from 0) and $(\nu\mathsf{G}, \mathit{out}) \cong \mathsf{Cofree}_{\mathsf{G}}\,1$.

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 sets of natural

transformations:

$$\mathscr{C}^{\mathscr{X}}(\mathsf{L}\circ\mathsf{F},\mathsf{G}) \cong \mathscr{D}^{\mathscr{X}}(\mathsf{F},\mathsf{R}\circ\mathsf{G}) \ , \tag{3.10a}$$

$$\mathscr{X}^{\mathscr{C}}(\mathsf{F}\circ\mathsf{R},\mathsf{G}) \cong \mathscr{X}^{\mathscr{D}}(\mathsf{F},\mathsf{G}\circ\mathsf{L}) \ . \tag{3.10b}$$

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

$$\lfloor \mathsf{H}f\cdot\sigma A\rfloor' = \tau B\cdot\mathsf{K}\lfloor f\rfloor \ , \tag{3.11a}$$

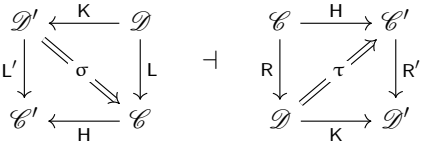
$$\mathsf{H}\lceil g\rceil\cdot\sigma A = \lceil \tau B\cdot\mathsf{K}g\rceil' \ , \tag{3.11b}$$

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 (3.10). In fact, each natural transformation uniquely determines the other:

$$\sigma A = \lceil \tau(\mathsf{L}A)\cdot\mathsf{K}(\eta A)\rceil' \ , \tag{3.12a}$$

$$\tau B = \lfloor \mathsf{H}(\epsilon B)\cdot\sigma(\mathsf{R}B)\rfloor' \ . \tag{3.12b}$$

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 (Palmquist, 1971).)

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

Adjunctions and Monads Huber (1961) 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.)

4 Warm-up: Mutumorphisms from Product Categories

Before we introduce the unified framework, it is instructive to walk through a specific instance. In Section 2 we mentioned that functions defined by mutual recursion, *mutumor-*

phisms, 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.)

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

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:

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

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 we have just calculated by expressing it as a diagram.

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

Note that B is an object of $\mathscr{C} \times \mathscr{C}$, that is, a pair of objects in $\mathscr{C}$, and recall that $\lfloor x \rfloor = x_1 \vartriangle x_2$.

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 (Fokkinga, 1992).

Several special cases are worth singling out. If x_2 does not depend on x_1, we obtain *zygomorphisms* (i.e. $b_2 := b_2 \cdot \mathsf{D}\ outr$ and consequently $x_2 := (\!|\, b_2 \,|\!)$). Further, when x_2 is the identity, the zygomorphism specialises to a paramorphism (i.e. $b_2 := in \cdot \mathsf{D}\ outr$ and

consequently $x_2 := (\!|in|\!) = id$). Pushing this to the extreme, if we have two independent homomorphisms (i.e. $b_1 := b_1 \cdot \mathsf{D}\ outl$ 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* (Bird & de Moor, 1997), 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 (Matthes & Uustalu, 2004) and *adjoint fold* in (Hinze, 2013), is not new. What is novel is the presentation, which makes essential use of conjugate pairs of distributive laws and liftings, rendering the proofs concise and elegant. Before we embark on our unification, we first take a short detour and explain some of the background that will help to connect the abstract concepts to concrete programs.

5.1 Background: Mendler-style Folds

Mendler-style folds (Mendler, 1991; Uustalu & Vene, 1999a) 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

$$\begin{aligned} \mathsf{depth}\ depth\ (\mathsf{Empty}) &= 0 \\ \mathsf{depth}\ depth\ (\mathsf{Node}\ l\ a\ r) &= 1 + (depth\ l\ \text{`}max\text{`}\ depth\ r) \ , \end{aligned}$$

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 ensures that the original recursion equation, $depth \cdot in = \mathsf{depth}\ depth$ has a *unique* solution—this is only 'roughly' the fixed point because of the occurrence of *in*.

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

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

where the base function Ψ is a natural transformation of type $\mathscr{C}(-,B) \dot{\to} \mathscr{C}(\mathsf{D}\ -, B)$. Our example is the special case where $x = depth$, when $\Psi = \mathsf{depth}$.

Very briefly, the Yoneda lemma (Mac Lane, 1998) 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. We therefore overload the notation for folds, where the unique solution to Equation (5.1) is written $(\!|\Psi|\!)$.

5.2 Background: 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.

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

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 $\mathsf{depths} \cdot (in \times \mathbb{N}) = \mathsf{depths}\ \mathit{depths}$ 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

$$x \cdot \mathsf{L}\ in = \Psi\ (\mu\mathsf{D})\ x \ , \tag{5.2}$$

where the base function $\Psi : \mathscr{C}(\mathsf{L}\ -, B) \dot{\to} \mathscr{C}(\mathsf{L}\ (\mathsf{D}\ -), B)$ is again a natural transformation. This time, our example is the special case where $x = \mathit{depths}$, when $\Psi = \mathsf{depths}$.

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, for instance, in *depths* we use the curry adjunction $- \times \mathbb{N} \dashv (-)^{\mathbb{N}}$; for others it is less obvious, for instance, for *total* the left adjoint is type application (applying a functor to a constant object), which has a right adjoint under some mild conditions (Hinze, 2013).

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' C.

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 $\mathsf{D} : \mathscr{D} \to \mathscr{D}$, a distributive law $\sigma : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L}$, and an algebra $b : \mathsf{C}\ B \to B$, an *adjoint recursion equation* in the unknown

$x : \mathsf{L}\ (\mu\mathsf{D}) \to B$ has the form

$$x \cdot \mathsf{L}\ in = b \cdot \mathsf{C}\ x \cdot \sigma\ (\mu\mathsf{D}) \quad . \tag{5.3}$$

The functor C is called a 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).

$$\mathsf{C} \circlearrowright \mathscr{C} \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{D} \circlearrowleft \mathsf{D} \tag{5.4}$$

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. To show that (5.3) 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

$$x \cdot \mathsf{L}\ a = b \cdot \mathsf{C}\ x \cdot \sigma\ A \quad , \tag{5.5}$$

and D-algebra homomorphisms. The key 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.

$$\begin{array}{ll}
 & x \cdot \mathsf{L}\ a = b \cdot \mathsf{C}\ x \cdot \sigma\ A \ :\ \mathsf{L}\ (\mathsf{D}\ A) \to B \\
\iff & \{\ \lfloor - \rfloor \text{ and } \lceil - \rceil \text{ are isomorphisms (3.3)}\ \} \\
 & \lfloor x \cdot \mathsf{L}\ a \rfloor = \lfloor b \cdot \mathsf{C}\ x \cdot \sigma\ A \rfloor \\
\iff & \{\ \lfloor - \rfloor \text{ is natural (3.4a)}\ \} \\
 & \lfloor x \rfloor \cdot a = \mathsf{R}\ b \cdot \lfloor \mathsf{C}\ x \cdot \sigma\ A \rfloor \\
\iff & \{\ \sigma \dashv \tau \text{ conjugates (3.11a)}\ \} \\
 & \lfloor x \rfloor \cdot a = \mathsf{R}\ b \cdot \tau\ B \cdot \mathsf{D}\ \lfloor x \rfloor \\
\iff & \{\ \text{definition of lifting (3.1a)}\ \} \\
 & \lfloor x \rfloor \cdot a = \mathsf{R}^{\tau}\ b \cdot \mathsf{D}\ \lfloor x \rfloor \ :\ \mathsf{D}\ A \to \mathsf{R}\ B
\end{array}$$

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.

$$\begin{array}{ccc}
\mathsf{L}\ (\mathsf{D}\ A) & \xrightarrow{\sigma\ A} & \mathsf{C}\ (\mathsf{L}\ A) \xrightarrow{\mathsf{C}\ x} \mathsf{C}\ B \\
{\scriptstyle \mathsf{L}\ a}\downarrow & & \downarrow{\scriptstyle b} \\
\mathsf{L}\ A & \xrightarrow[x]{} & B
\end{array}
\quad \iff \quad
\begin{array}{ccc}
\mathsf{D}\ A & \xrightarrow{\mathsf{D}\ \lfloor x \rfloor} & \mathsf{D}\ (\mathsf{R}\ B) \\
{\scriptstyle a}\downarrow & & \downarrow{\scriptstyle \mathsf{R}^{\tau}\ b} \\
A & \xrightarrow[\lfloor x \rfloor]{} & \mathsf{R}\ B
\end{array} \tag{5.6}$$

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

Theorem 5.2 (*Adjoint folds*)
Given an adjunction $\mathsf{L} \dashv \mathsf{R} : \mathscr{C} \rightharpoonup \mathscr{D}$, functors $\mathsf{C} : \mathscr{C} \to \mathscr{C}$ and $\mathsf{D} : \mathscr{D} \to \mathscr{D}$, a distributive law $\sigma : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L}$ with conjugate $\tau : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C}$, and an algebra $b : \mathsf{C}\ B \to B$, then the adjoint recursion equation (5.3) in the unknown $x : \mathsf{L}\ (\mu\mathsf{D}) \to B$ is

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

and has the unique solution $x = \lceil (\!| \mathsf{R}^{\tau}\ b |\!) \rceil$. The arrow x is called an *adjoint fold*.

Proof
This is an immediate consequence of initiality.

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

□

So an adjoint fold is a traho from the initial algebra. Using the bijection (5.6) 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:

$$\begin{array}{ccc}
\mathsf{C}\text{-}\mathbf{Alg}(\mathscr{C}) & \xrightarrow{\mathsf{R}^{\tau}} & \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D}) \\
\downarrow{\scriptstyle \mathsf{U}^{\mathsf{C}}} & & \downarrow{\scriptstyle \mathsf{U}^{\mathsf{D}}} \\
\mathsf{C} \circlearrowright\ \mathscr{C} & \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} & \mathscr{D}\ \circlearrowleft \mathsf{D}
\end{array} \tag{5.7}$$

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

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 5.6. But first, let us look at an example.

Example 5.3 (*Mutumorphisms*)
Mutumorphisms are an instance of adjoint folds where the adjunction involved is $\Delta \dashv (\times)$, the control functor is $\Delta \circ \mathsf{D} \circ (\times)$, and $\sigma = \Delta \circ \mathsf{D} \circ \eta$.

$$\Delta \circ \mathsf{D} \circ (\times) \circlearrowright\ \mathscr{D}^2 \ \underset{(\times)}{\overset{\Delta}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}}\ \mathscr{D}\ \circlearrowleft \mathsf{D}$$

The conjugate of σ is $\tau = \eta \circ \mathsf{D} \circ (\times)$ (3.12b) and thus

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

Note that the lifted product functor $(\times)^{\tau}$ is just the left adjunct. Looking more closely, one might notice that once the adjunction was established, very few choices needed to be made: the choice of control functor and its associated conjugates were all deeply connected. This motivates the introduction of a *canonical* adjoint fold in the next section.

5.4 Canonical Adjoint Folds

Adjoint folds involve several pieces of data: an adjunction, an algebra, and a control functor equipped with a conjugate pair of distributive laws. 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 (5.4). 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 a canonical choice for the conjugate too:

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

For this case the development of adjoint folds in Section 5.3 can be simplified.

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 $\lfloor b \rfloor = \mathsf{R}^{\tau}\, b$ and $\mathsf{L}\, (\mathsf{D}\, \lfloor x \rfloor) = \mathsf{C}\, x \cdot \sigma\, A$.

$$\begin{array}{ccc} \mathsf{L}\,(\mathsf{D}\,A) & \xrightarrow{\mathsf{L}\,(\mathsf{D}\,\lfloor x \rfloor)} & \mathsf{L}\,(\mathsf{D}\,(\mathsf{R}\,B)) \\ {\scriptstyle \mathsf{L}\,a}\downarrow & & \downarrow{\scriptstyle b} \\ \mathsf{L}\,A & \xrightarrow[x]{} & B \end{array} \qquad \Longleftrightarrow \qquad \begin{array}{ccc} \mathsf{D}\,A & \xrightarrow{\mathsf{D}\,\lfloor x \rfloor} & \mathsf{D}\,(\mathsf{R}\,B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle \lfloor b \rfloor} \\ A & \xrightarrow[\lfloor x \rfloor]{} & \mathsf{R}\,B \end{array} \tag{5.8}$$

Now, if (a, A) is initial, then $x = \lceil (\!| \lfloor b \rfloor |\!) \rceil$. This gives us the following definition.

Definition 5.4 (*Canonical adjoint recursion equation*)

Given an adjunction $\mathsf{L} \dashv \mathsf{R} : \mathscr{C} \rightharpoonup \mathscr{D}$, a functor $\mathsf{D} : \mathscr{D} \to \mathscr{D}$, and an algebra $b : (\mathsf{L} \circ \mathsf{D} \circ \mathsf{R})\, B \to B$, a *canonical adjoint recursion equation* in the unknown $x : \mathsf{L}\, (\mu \mathsf{D}) \to B$ has the form

$$x \cdot \mathsf{L}\, in = b \cdot (\mathsf{L} \circ \mathsf{D} \circ \mathsf{R})\, x \cdot (\mathsf{L} \circ \mathsf{D} \circ \eta)\, (\mu \mathsf{D}) \;\; . \tag{5.9}$$

The adjective ‘canonical’ needs some justification. We show that every other choice of control functor and conjugates can be reduced to the canonical one. Assume that we have another control functor C', and a pair of conjugate distributive laws

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

Using bijection (3.10b), 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 γ-**Alg**$(\mathscr{C})$, which maps C'-algebras to C-algebras. Since it is a lifting of the identity functor, γ-**Alg**$(\mathscr{C})$ is faithful. Moreover, we have the following commutative

diagrams of functors.

$$\begin{array}{ccc} \mathsf{C}\text{-}\mathbf{Alg}(\mathscr{C}) & \xrightarrow{\mathsf{R}^{\tau}} & \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D}) \\ {\scriptstyle \gamma\text{-}\mathbf{Alg}(\mathscr{C})}\uparrow & & \| \\ \mathsf{C}'\text{-}\mathbf{Alg}(\mathscr{C}) & \xrightarrow[\mathsf{R}^{\tau'}]{} & \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D}) \end{array} \tag{5.10}$$

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

$$\sigma' = \gamma \circ \mathsf{L} \cdot \sigma \tag{5.11a}$$

$$\tau' = \mathsf{R} \circ \gamma \cdot \tau \tag{5.11b}$$

For the proof of (5.10) 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:

$$\begin{array}{ll} & x \cdot \mathsf{L}\ a = b \cdot \mathsf{C}'\ x \cdot \sigma'\ A \\ \iff & \{\ (5.11a)\ \} \\ & x \cdot \mathsf{L}\ a = b \cdot \mathsf{C}'\ x \cdot \gamma\ (\mathsf{L}\ A) \cdot \sigma\ A \\ \iff & \{\ \gamma \text{ is natural and } x : \mathsf{L}\ A \to B\ \} \\ & x \cdot \mathsf{L}\ a = b \cdot \gamma\ B \cdot \mathsf{C}\ x \cdot \sigma\ A \\ \iff & \{\ \text{definition of lifting (3.1b)}\ \} \\ & x \cdot \mathsf{L}\ a = \gamma\text{-}\mathbf{Alg}(\mathscr{C})\ b \cdot \mathsf{C}\ x \cdot \sigma\ A \ . \end{array}$$

This result tells us that we only need a canonical adjoint fold, and that uniqueness follows from this alone.

5.5 From Mendler-style to Canonical Adjoint Folds

Recursive Haskell programs are easily framed as Mendler-style adjoint folds (5.2). Adjoint folds (5.3) are, however, preferable for the theoretical development as they avoid sophistications such as natural transformations between hom-functors. Nevertheless, Mendler-style adjoint folds are useful in their own right: they make the translation from definitions with explicit recursion relatively simple. To illustrate the difference between the two styles, let us consider how *depths*, which we defined with a Mendler-style adjoint fold in Section 5.2 might be rendered using a canonical adjoint fold.

The adjunction has already been identified as $- \times \mathbb{N} \dashv (-)^{\mathbb{N}}$, which is the curry adjunction. Thus, using a canonical adjoint fold, we choose the control functor to be $- \times \mathbb{N} \circ \mathsf{Tree} \circ (-)^{\mathbb{N}}$, we need only supply an algebra depths:

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

Showing that this results in the definition of *depths*, however, requires us to work with the distributive law, as well as the action of the canonical functor on the recursion equation. In

this case, we have $\eta = pair$, where $pair\ x\ y = (x,y)$, and we recall that the actions on arrows for our functors are $(-)^{\mathbb{N}}\ f = (f\cdot)$ and $(-\times\mathbb{N})\ f = f\times id$. Putting this together, we have:

$$\begin{aligned} depths\cdot(in\times id) &= \mathsf{depths}\cdot(-\times\mathbb{N}\circ\mathsf{Tree})\ ((-)^{\mathbb{N}}\ depths\cdot pair) \\ &= \mathsf{depths}\cdot(\mathsf{Tree}\ ((depths\ \cdot)\cdot pair)\times id) \end{aligned}$$

By doing case analysis, we do indeed recover the original definition: the *Empty* case falls out almost immediately, and we can calculate the case for *Node*:

$$\begin{aligned} depths\ (Node\ l\ a\ r,n) &= \mathsf{depths}\ (Node\ (((depths\ \cdot)\cdot pair)\ l)\ a\ (((depths\ \cdot)\cdot pair)\ r),n) \\ &= Node\ (((depths\ \cdot)\cdot pair)\ l\ (n+1))\ n\ (((depths\ \cdot)\cdot pair)\ r\ (n+1)) \\ &= Node\ (depths\ (l,n+1))\ n\ (depths\ (r,n+1) \end{aligned}$$

While we have successfully retrieved the definition of *depths*, even for this simple example the process is not as straight-forward as the Mendler-style adjoint fold we saw in Section 5.2.

As in the vanilla case, Mendler-style adjoint folds (5.2) and adjoint folds (5.3) are interchangeable. Every adjoint fold is a Mendler-style one, since the right-hand side of (5.3) 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 $b : \mathsf{C}\ B \to B$. Since we have an adjunction at our disposal, we can choose the canonical control functor and distributive law. All that is left is the C-algebra, which we can derived from the base function: $b = \Psi\ (\mathsf{R}\ B)\ (\epsilon\ B) : \mathsf{L}\ (\mathsf{D}\ (\mathsf{R}\ B)) \to B$. To prove that $\Psi\ X\ x = b\cdot \mathsf{C}\ x\cdot \sigma\ X$ we reason as follows:

$$\begin{aligned} & b\cdot \mathsf{C}\ x\cdot \sigma\ X \\ = & \quad \{\ \text{definition of } b, \mathsf{C}, \text{and } \sigma\ X\ \} \\ & \Psi\ (\mathsf{R}\ B)\ (\epsilon\ B)\cdot \mathsf{L}\ (\mathsf{D}\ (\mathsf{R}\ x))\cdot \mathsf{L}\ (\mathsf{D}\ (\eta\ X)) \\ = & \quad \{\ \text{functoriality of } \mathsf{L} \text{ and } \mathsf{D}\ \} \\ & \Psi\ (\mathsf{R}\ B)\ (\epsilon\ B)\cdot \mathsf{L}\ (\mathsf{D}\ (\mathsf{R}\ x\cdot \eta\ X)) \\ = & \quad \{\ \text{naturality of } \Psi\ \} \\ & \Psi\ X\ (\epsilon\ B\cdot \mathsf{L}\ (\mathsf{R}\ x\cdot \eta\ X)) \\ = & \quad \{\ \text{functoriality of } \mathsf{L}\ \} \\ & \Psi\ X\ (\epsilon\ B\cdot \mathsf{L}\ (\mathsf{R}\ x)\cdot \mathsf{L}\ (\eta\ X)) \\ = & \quad \{\ \text{naturality of } \epsilon\ \} \\ & \Psi\ X\ (x\cdot \epsilon\ (\mathsf{L}\ X)\cdot \mathsf{L}\ (\eta\ X)) \\ = & \quad \{\ \text{triangle identity (3.5a)}\ \} \\ & \Psi\ X\ x \end{aligned}$$

On the surface, the canonical control functor and its associated conjugates are rather mysterious, especially if we try to link the Haskell programs in Section 2 *directly* to the recursion scheme of adjoint folds (5.3). This calculation shows how canonical adjoint folds connect directly to the more familiar Mendler-style equations.

5.6 Restoring Symmetry

Let us now assume that the distributive law σ is an isomorphism. When this is the case we can continue the first calculation of Section 5.3 'in the opposite direction'. We start

with (5.5) and reason

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

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

$$\begin{array}{ccc} \mathsf{C}\,(\mathsf{L}\,A) & \xrightarrow{\mathsf{C}\,x} & \mathsf{C}\,B \\ {\scriptstyle \mathsf{L}^{\sigma^\circ}\,a}\downarrow & & \downarrow{\scriptstyle b} \\ \mathsf{L}\,A & \xrightarrow[x]{} & B \end{array} \quad \overset{\sigma \text{ iso}}{\iff} \quad \begin{array}{ccc} \mathsf{D}\,A & \xrightarrow{\mathsf{D}\,\lfloor x \rfloor} & \mathsf{D}\,(\mathsf{R}\,B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle \mathsf{R}^\tau\,b} \\ A & \xrightarrow[\lfloor x \rfloor]{} & \mathsf{R}\,B \end{array} \tag{5.12}$$

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 (5.7) with the missing left adjoints.

$$\begin{array}{ccc} \mathsf{C}\text{-}\mathbf{Alg}(\mathscr{C}) & \underset{\mathsf{R}^\tau}{\overset{\mathsf{L}^{\sigma^\circ}}{\leftrightarrows}} & \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D}) \\ {\scriptstyle \mathsf{Free}^{\mathsf{C}}} \uparrow \dashv \downarrow {\scriptstyle \mathsf{U}^{\mathsf{C}}} & \cong & {\scriptstyle \mathsf{Free}^{\mathsf{D}}} \uparrow \dashv \downarrow {\scriptstyle \mathsf{U}^{\mathsf{D}}} \\ \mathsf{C} \circlearrowright\ \mathscr{C} & \underset{\mathsf{R}}{\overset{\mathsf{L}}{\leftrightarrows}} & \mathscr{D}\ \circlearrowright \mathsf{D} \end{array} \tag{5.13}$$

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

Overall, we have four (!) adjunctions, which form a commuting square of adjunctions. The proof of this fact makes use of the high-level reasoning principle (3.7). If we instantiate (3.7) to the compositions of left and right adjoints (note that left adjoints are composed in the opposite order) we obtain:

$$\mathsf{L}^{\sigma^\circ} \circ \mathsf{Free}^{\mathsf{D}} \cong \mathsf{Free}^{\mathsf{C}} \circ \mathsf{L} \quad \overset{\sigma \text{ iso}}{\iff} \quad \mathsf{U}^{\mathsf{D}} \circ \mathsf{R}^\tau \cong \mathsf{R} \circ \mathsf{U}^{\mathsf{C}}\ .$$

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 5.5
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.

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

where F^* is the free monad for an endofunctor F, defined at the end of Section 3.3.

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

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

□

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.* (1995)'s categorical fixed-point calculus.

Corollary 5.6 (*Type fusion*)
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.

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

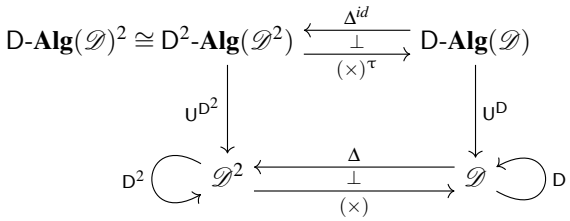
Example 5.7
The diagonal functor Δ satisfies a simple property: $\Delta \circ \mathsf{D} = \mathsf{D}^2 \circ \Delta$. Since Δ is a left adjoint, Corollary 5.6 implies

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

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 $\tau = \mathsf{D}\ \mathit{outl} \vartriangle \mathsf{D}\ \mathit{outr}$ (3.12b) and thus

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

Instantiating Diagram (5.13) we can see the global picture.



Since D^2-**Alg**$(\mathscr{D}^2) \cong \mathsf{D}$-**Alg**$(\mathscr{D})^2$, we obtain that $(\times)^\tau$ modulo the isomorphism is the product functor for D-**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))

Example 5.8
We have now encountered two control functors associated with the adjunction $\Delta \dashv (\times)$: mutumorphisms are based on the canonical control functor $\mathsf{C} = \Delta \circ \mathsf{D} \circ (\times)$; banana-split employs the 'perfect' control functor D^2. In Section 4 we noted that the banana-split law (4.1) arises as an extreme case of mutumorphisms. We can relate the two by showing the appropriate lifting.

The lifting γ-**Alg**$(\mathscr{C}^2) : \mathsf{D}^2$-**Alg**$(\mathscr{C}^2) \to \mathsf{C}$-**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-algebras into C-algebras.

6 Detour: Zygomorphisms from Slice Categories

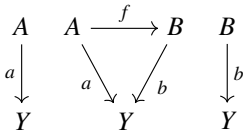
In Section 4, we saw that zygomorphisms can be expressed as a special case of mutumorphisms. However, we can also express zygomorphisms directly in terms of an adjoint fold using the adjunction that exists in the construction of slice categories.

A zygomorphism is essentially a recursion equation that depends on some auxiliary recursion equation over the same data structure. A slice category adorns objects in its base category with an arrow that points to some object of interest. In our setting, this allows us to carry around information about the auxiliary function. In fact, this technique of using a slice category has been used to give a semantics to generic functions by the authors in (Hinze & Wu, 2011); much of the background required is identical, and we repeat it here.

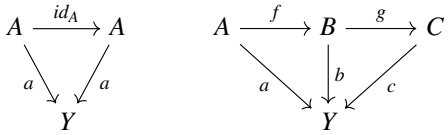
6.1 Background: Slice Categories

Slice categories are a construction that involve the objects and arrows of some base category. The fact that they resemble categories of coalgebras should be no surprise, since they are a special case where the functor is constant.

Slice Categories Let $\mathscr{C}$ be a category and let $Y : \mathscr{C}$ be an object of $\mathscr{C}$. An object of the *slice category* $\mathscr{C} \downarrow Y$ is a pair (A,a) where $A : \mathscr{C}$ is an object and $a : A \to Y : \mathscr{C}$ is an arrow. An arrow $f : (A,a) \to (B,b) : \mathscr{C} \downarrow Y$ of the slice category is an arrow $f : A \to B : \mathscr{C}$ of the underlying category such that $a = b \cdot f$.



In short, objects are arrows and arrows are commuting triangles. Identity and composition are inherited from the base category $\mathscr{C}$. Clearly, id_A serves as the identity on (A,a) as $a = a \cdot id_A$. The diagram below shows that composition takes commuting triangles to commuting triangles: $b = c \cdot g$ and $a = b \cdot f$ imply $a = c \cdot g \cdot f$.



A slice category adds structure on top of a base category. In such a situation, there is a functor that forgets about the extra structure. The *forgetful* or *underlying functor* $\mathsf{U}_Y : \mathscr{C} \downarrow Y \to \mathscr{C}$ forgets about the base object Y and the arrows into Y:

$$\begin{aligned} \mathsf{U}_Y\,(A,a) &= A \ , \\ \mathsf{U}_Y f &= f \ . \end{aligned}$$

If the category $\mathscr{C}$ has products, then the forgetful functor U_Y has a right adjoint, the so-called *pairing functor* $\mathsf{P}_Y : \mathscr{C} \to \mathscr{C}\downarrow Y$.

$$\mathscr{C} \underset{\mathsf{P}_Y}{\overset{\mathsf{U}_Y}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C}\downarrow Y \tag{6.1}$$

The pairing functor is defined

$$\begin{aligned} \mathsf{P}_Y A &= (A \times Y, \mathit{outr}) \ , \\ \mathsf{P}_Y f &= f \times Y \ . \end{aligned}$$

The functor P_Y pairs its argument with Y, hence its name. It respects the types, $\mathsf{P}_Y f \colon \mathsf{P}_Y A \to \mathsf{P}_Y B$, as $\mathit{outr} = \mathit{outr} \cdot (f \times Y)$. To establish the adjunction we have to show that certain arrows in $\mathscr{C}$ are in one-to-one correspondence with certain arrows in $\mathscr{C}\downarrow Y$:

$$\mathscr{C}(\mathsf{U}_Y\,(A,a),B) \cong (\mathscr{C}\downarrow Y)((A,a),\mathsf{P}_Y\,B) \ .$$

Intuitively the adjunction captures the idea of *caching*: an attribute $a : A \to Y$ is cached by pairing B with a's value. The adjuncts make this explicit

$$\begin{aligned} \lfloor f \colon \mathsf{U}_Y\,(A,a) \to B \rfloor &= f \vartriangle a \ , \\ \lceil g : (A,a) \to \mathsf{P}_Y\,B \rceil &= \mathit{outl} \cdot g \ . \end{aligned}$$

The left adjunct respects the types, $\lfloor f \rfloor : (A,a) \to \mathsf{P}_Y\,B$, as $a = \mathit{outr} \cdot (f \vartriangle a)$. We leave it as an exercise to prove that these functions are indeed inverses.

Liftings Given a functor $\mathsf{F} : \mathscr{C} \to \mathscr{D}$, we say that the functor $\bar{\mathsf{F}} : \mathscr{C}\downarrow Y \to \mathscr{D}\downarrow Z$ is a lifting if there is the equivalence

$$\mathsf{U}_Z \circ \bar{\mathsf{F}} = \mathsf{F} \circ \mathsf{U}_Y \ . \tag{6.2}$$

Such liftings are known to be in a one-to-one correspondance with natural transformations of the type

$$\kappa : \forall A \,.\, \mathscr{C}\,(A,Y) \to \mathscr{D}\,(\mathsf{F}\,A,Z) \ . \tag{6.3}$$

Given such a natural transformation, the lifting $\mathsf{F}_\kappa : \mathscr{C}\downarrow Y \to \mathscr{D}\downarrow Z$ is given by:

$$\begin{aligned} \mathsf{F}_\kappa\,(A,a) &= (\mathsf{F}\,A, \kappa\,A\,a) \ , \\ \mathsf{F}_\kappa f &= \mathsf{F} f \ . \end{aligned}$$

The action on A and f are given by F and since this is functorial it follows that F_κ preserves identity and composition. The only difficulty in showing that this is a functor lies in establishing that κ respects the types. We must show that

$$f \colon (A,a) \to (B,b) \Rightarrow \mathsf{F}_\kappa f \colon \mathsf{F}_\kappa\,(A,a) \to \mathsf{F}_\kappa\,(B,b) \ .$$

We reason

$$
\begin{array}{ll}
 & \kappa\, B\, b \cdot \mathsf{F}\, f \\
= & \{\ \kappa \text{ is natural } \} \\
 & \kappa\, A\, (b \cdot f) \\
= & \{\ \text{assuming } f\colon (A,a) \to (B,b)\ \} \\
 & \kappa\, A\, a\ .
\end{array}
$$

The fact that F_κ is a lifting follows directly from the definition. On objects we have:

$$\mathsf{U}_Z\,(\mathsf{F}_\kappa\,(A,a)) = \mathsf{U}_Z\,(\mathsf{F}\,A, \kappa\,A\,a) = \mathsf{F}\,A = \mathsf{F}\,(\mathsf{U}_Z\,(A,a))\ ,$$

and on arrows:

$$\mathsf{U}_Z\,(\mathsf{F}_\kappa f) = \mathsf{U}_Z\,(\mathsf{F} f) = \mathsf{F} f = \mathsf{F}\,(\mathsf{U}_Z f)\ .$$

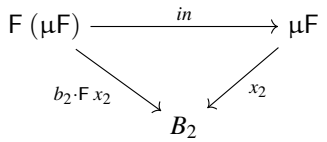
6.2 Zygomorphisms Revisited

A zygomorphism is a pair of morphisms $x_1 : \mu\mathsf{F} \to B_1$ and $x_2 : \mu\mathsf{F} \to B_2$, where the definition of x_1 depends on x_2. The equations rely on algebras $b_1 : \mathsf{F}\,(B_1 \times B_2) \to B_1$ and $b_2 : \mathsf{F}\,B_2 \to B_2$, and are the solution to the following pair of equations:

$$x_1 \cdot in = b_1 \cdot \mathsf{F}\,(x_1 \vartriangle x_2) \ \text{ and } \ x_2 \cdot in = b_2 \cdot \mathsf{F}\,x_2\ . \tag{6.4}$$

Our task is to show how we can express these two equations as a catamorphism in the slice category.

The definition of x_1 relies on the existence of x_2, and so intuitively we will need information to recover x_2. The definition of x_2 can be depicted by the following commuting triangle:



This motivates us to work with the slice category $\mathscr{C}\downarrow B_2$. The underlying functor that we have to start with is $\mathsf{F} : \mathscr{C} \to \mathscr{C}$ and we will lift this to a functor on the slice category where $\bar{\mathsf{F}} := \mathsf{F}_\kappa$ is given by a natural transformation $\kappa : \forall A\ .\ \mathscr{C}\,(A,B_2) \to \mathscr{C}\,(\mathsf{F}\,A,B_2)$. Since we have b_2 at our disposal, the following suggests itself:

$$\kappa\,A\,(h : A \to B_2) = b_2 \cdot \mathsf{F}\,h\ . \tag{6.5}$$

This definition conveniently gives us the equality

$$\bar{\mathsf{F}}\,(\mu\mathsf{F}, x_2) = (\mathsf{F}\,(\mu\mathsf{F}), b_2 \cdot \mathsf{F}\,x_2)\ , \tag{6.6}$$

and this corresponds to the left-hand-side of the commuting triangle above.

Now we can investigate the result of applying the canonical adjoint fold to the functor $\bar{\mathsf{F}}$. Instantiating Equation (5.8) with this data gives us:

$$\begin{array}{ccc} \mathsf{U}_{B_2}\,(\bar{\mathsf{F}}\,(\mu\mathsf{F}, x_2)) & \xrightarrow{\mathsf{U}_{B_2}\,(\bar{\mathsf{F}}\,\lfloor x \rfloor)} & \mathsf{U}_{B_2}\,(\bar{\mathsf{F}}\,(\mathsf{P}_{B_2}\,B)) \\ {\scriptstyle \mathsf{U}_{B_2}\,in}\downarrow & & \downarrow{\scriptstyle b_1} \\ \mathsf{U}_{B_2}\,((\mu\mathsf{F}, x_2)) & \xrightarrow[x]{} & B \end{array} \quad \Longleftrightarrow \quad \begin{array}{ccc} \bar{\mathsf{F}}\,(\mu\mathsf{F}, x_2) & \xrightarrow{\bar{\mathsf{F}}\,\lfloor x \rfloor} & \bar{\mathsf{F}}\,(\mathsf{P}_{B_2}\,B) \\ {\scriptstyle in}\downarrow & & \downarrow{\scriptstyle \lfloor b_1 \rfloor} \\ (\mu\mathsf{F}, x_2) & \xrightarrow[\lfloor x \rfloor]{} & \mathsf{P}_{B_2}\,B \end{array} \tag{6.7}$$

The diagram on the right is a catamorphism in the slice category. On the left hand side, we can unpack the definitions to reveal that $x \cdot in = b_1 \cdot \mathsf{F}\,(x \vartriangle x_2)$ and this is precisely the defining equation for the zygomorphism x_1.

7 Unifying Recursion Schemes from Comonads

Recursion schemes from comonads (Uustalu *et al.*, 2001), *rsfcs* for short, form a general recursion principle that makes use of a comonad N to provide 'contextual information' to the algebra. This scheme covers a wide range of different morphisms from the zoo, but not all of them: nested datatypes are an example. In this section we will show how rsfcs are in fact an instance of adjoint folds.

First, we introduce the relevant background to understand the construction of an rsfc in Section 7.1. The formal definition of an rsfc is then given in Section 7.2. We introduce Eilenberg-Moore categories in Section 7.3, and bialgebras in Section 7.4 since these concepts will be required to prove that rsfcs have a unique solution. This uniqueness is shown in Section 7.5 where an adjunction to the Eilenberg-Moore category is used to instantiate an adjoint fold.

7.1 Background: Comonads and Distributive Laws

Recursion schemes from monads make use of two concepts: comonads and distributive laws over comonads. These concepts are presented in this section.

Comonads 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:

$$\epsilon \circ \mathsf{N} \cdot \delta = \mathsf{N} \ , \tag{7.1a}$$

$$\mathsf{N} \circ \epsilon \cdot \delta = \mathsf{N} \ , \tag{7.1b}$$

$$\delta \circ \mathsf{N} \cdot \delta = \mathsf{N} \circ \delta \cdot \delta \ . \tag{7.1c}$$

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.

Distributive laws over comonads 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:

$$\epsilon \circ \mathsf{F} \cdot \lambda = \mathsf{F} \circ \epsilon \ , \tag{7.2a}$$

$$\delta \circ \mathsf{F} \cdot \lambda = \mathsf{N} \circ \lambda \cdot \lambda \circ \mathsf{N} \cdot \mathsf{F} \circ \delta \ . \tag{7.2b}$$

Distributing the comonad outside the functor and focusing is the same as focusing inside the functor. Distributing the comonad outside the functor and duplicating is the same as duplicating inside the functor, and then distributing.

7.2 Recursion schemes from comonads

Just as with an adjoint fold, an rsfc is 'doubly generic': it is parametric in a datatype $\mu\mathsf{F}$, and in a 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 (Uustalu & Vene, 2011), but this is outside the scope of this paper.) To this end it makes use of a coalgebra $fan : \mu\mathsf{F} \to \mathsf{N}\ (\mu\mathsf{F})$ 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 $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$ as detailed below.

Definition 7.1 (*Comonadic recursion equation*)
Given a functor F, a comonad $(\mathsf{N}, \epsilon, \delta)$, a distributive law $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$, and an algebra $b : \mathsf{F}\ (\mathsf{N}\ B) \to B$, a comonadic recursion equation in the unknown $f : \mu\mathsf{F} \to B$ has the form

$$f \cdot in = b \cdot \mathsf{F}\ (\mathsf{N}\ f \cdot fan) \ , \tag{7.3}$$

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

The composition $\mathsf{N}\ f \cdot fan$ 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.* (2001) showed the following

Theorem 7.2 (*Rsfcs*)
The comonadic recursion equation (7.3) 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 7.2 to Section 7.5, after we have provided the necessary background in Sections 7.3 and 7.4, which can be skipped by those already familiar with the material.

When the cofree comonad is used to provide the contextual information for an rsfc, the ensuing recursion scheme is a histomorphism (Uustalu *et al.*, 2001). In Haskell, we can

represent the cofree comonad for a functor as:

$$\textbf{data}\ \mathsf{G}_\infty\ a = \mathit{Cons}_\infty\ \{\mathit{head}_\infty :: a, \mathit{tail}_\infty :: \mathsf{G}\ (\mathsf{G}_\infty\ a)\}$$

For the definition of the counit, we simply have $\epsilon = \mathit{head}_\infty$. The comultiplication can be given in terms of the so-called *trace* function, written $\llbracket\text{-}\rrbracket$. This uses a coalgebra to corecursively build a new level of the structure from a seed.

$$\begin{aligned}&\llbracket\text{-}\rrbracket :: \mathit{Functor}\ \mathsf{G} \Rightarrow (c \to \mathsf{G}\ c) \to c \to \mathsf{G}_\infty\ c\\ &\llbracket c \rrbracket = h\ \textbf{where}\ h = \mathit{cons}_\infty \cdot (\mathit{id} \vartriangle \mathit{fmap}\ h \cdot c)\end{aligned}$$

A given seed is used to both populate the outer level of the structure, and as an argument to the coalgebra that unpacks a new level of seeds, to which the trace is corecursively applied. We make use of cons_∞, the uncurried version of Cons_∞ to assemble these two parts. In effect, this is the *fan* that corresponds to a cofree comonad. The comultiplication is then simply $\delta = \llbracket \mathit{tail}_\infty \rrbracket$, where the seed is a cofree comonad, and the tail of its structure is recursively embedded. The trace can also be used to define a distributive law:

$$\begin{aligned}&\lambda :: \mathit{Functor}\ \mathsf{G} \Rightarrow \mathsf{G}\ (\mathsf{G}_\infty\ a) \to \mathsf{G}_\infty\ (\mathsf{G}\ a)\\ &\lambda = \mathit{fmap}\ (\mathit{fmap}\ \mathit{head}_\infty) \cdot \llbracket \mathit{fmap}\ \mathit{tail}_\infty \rrbracket\end{aligned}$$

This gives us all the ingredients we need to construct a histomorphism from an rsfc.

Example 7.3
The *knapsack* function is an example of a histomorphism, where the underlying functor is Nat, with constructors Zero and Succ. The algebra is given by $\mathsf{knapsack}$, where we assume that $\mathit{wvs} :: [(\mathbb{N}, \mathbb{R})]$ is implicitly supplied.

$$\begin{aligned}&\mathsf{knapsack} :: \mathsf{Nat}\ (\mathsf{Nat}_\infty\ \mathbb{R}) \to \mathbb{R}\\ &\mathsf{knapsack}\ \mathsf{Zero} = 0\\ &\mathsf{knapsack}\ (\mathsf{Succ}\ \mathit{table}) =\\ &\quad \mathit{maximum}_0\ [v+u \mid (w+1, v) \leftarrow \mathit{wvs}, \mathit{Just}\ u \leftarrow [\mathit{lookup}_\infty\ \mathit{table}\ w]]\end{aligned}$$

This makes use of an auxiliary function lookup_∞ that takes a table of type $\mathsf{Nat}_\infty\ a$ of values indexed by natural numbers.

$$\begin{aligned}&\mathit{lookup}_\infty :: \mathsf{Nat}_\infty\ a \to \mathbb{N} \to \mathit{Maybe}\ a\\ &\mathit{lookup}_\infty\ (\mathit{Cons}_\infty\ a\ m)\ 0 = \mathit{Just}\ a\\ &\mathit{lookup}_\infty\ (\mathit{Cons}_\infty\ a\ \mathsf{Zero})\ (n+1) = \mathit{Nothing}\\ &\mathit{lookup}_\infty\ (\mathit{Cons}_\infty\ a\ (\mathsf{Succ}\ \mathit{table}))\ (n+1) = \mathit{lookup}_\infty\ \mathit{table}\ n\end{aligned}$$

This definition is given by induction on the naturals. In the base case, we simply retrieve the value at the head of table. Otherwise, we look deeper into the table: if the tail is empty, then the result is *Nothing*, since there are no more values to look up, otherwise, we recurse deeper into the table.

Putting the pieces together, we can obtain the following definition of *knapsack*:

$$\begin{aligned}&\mathit{knapsack} :: \mathbb{N} \to \mathbb{R}\\ &\mathit{knapsack} = \mathit{head}_\infty \cdot (\!|\mathit{fmap}\ \mathsf{knapsack} \cdot \lambda \cdot \mathit{fmap} \llbracket \mathit{tail}_\infty \rrbracket|\!)\end{aligned}$$

While this construction corresponds to the interpretation of a histomorphism as an rsfc, it suffers from the fact that the body of the fold involves applying the algebra to the result of a

trace: its behaviour is quadratic. A little more work is required to derive a linear version, but the details are beyond the scope of this paper and covered fully by Hinze & Wu (2013). In short, we can collapse this into a fold that invokes the algebra only once per level.

7.3 Background: Eilenberg-Moore Categories

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

$$\epsilon\ C \cdot c = id_C \ , \tag{7.4a}$$

$$\delta\ C \cdot c = \mathsf{N}\ c \cdot c \ . \tag{7.4b}$$

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 (7.1b) and (7.1c). The second law (7.4b) 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: G-**Coalg**$(\mathscr{C}) \cong \mathscr{C}_\mathsf{N}$ where $\mathsf{N} = \mathsf{G}_\infty$ is the cofree comonad.

Distributive laws and liftings 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 $\mathsf{N}^\lambda : \mathsf{F}$-**Alg**$(\mathscr{C}) \to \mathsf{F}$-**Alg**$(\mathscr{C})$. Now, the coherence conditions ensure that N^λ is a comonad with $\epsilon_{a,A} = \epsilon\ A$ and $\delta_{a,A} = \delta\ A$. In particular, the lifted transformations $\epsilon : \mathsf{N}^\lambda \dot{\to} \mathsf{Id}$ and $\delta : \mathsf{N}^\lambda \dot{\to} \mathsf{N}^\lambda \circ \mathsf{N}^\lambda$ are F-algebra homomorphisms.

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 (1965), the other by Eilenberg & Moore (1965). Here we 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:

$$\mathsf{Cofree}_\mathsf{N}\ B = (\mathsf{N}\ B, \delta\ B) \ , \tag{7.5a}$$

$$\mathsf{Cofree}_\mathsf{N}\ f = \mathsf{N}\ f \ . \tag{7.5b}$$

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 $(C,c) \to (\mathsf{N}\ C, \delta\ C)$ (7.4b). 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 $f\colon \mathsf{U_N}\ (C,c) \to B$ and $h : (C,c) \to \mathsf{Cofree_N}\ B$. The adjunction $\mathsf{U_N} \dashv \mathsf{Cofree_N}$ indeed generates the comonad N: we have $\mathsf{U_N} \circ \mathsf{Cofree_N} = \mathsf{N}$ and $\delta = \mathsf{U_N} \circ \eta \circ \mathsf{Cofree_N}$. Since $\mathsf{U_N}$ is faithful, we can simplify the bijection slightly:

$$f = \epsilon\ B \cdot h \quad \Longleftrightarrow \quad \mathsf{N}\, f \cdot c = h \ , \tag{7.6}$$

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?

7.4 Background: Bialgebras

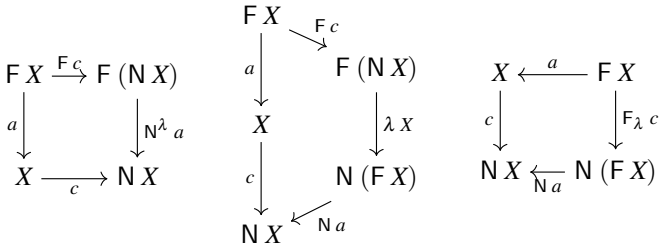
A bialgebra combines an algebra and a coalgebra with a common carrier. Bialgebras come in many flavours; we need the variant that combines F-algebras and coalgebras for a comonad N. The two functors have to interact coherently, described by a distributive law.

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:

$$c \cdot a = \mathsf{N}\, a \cdot \lambda\ X \cdot \mathsf{F}\, c \ . \tag{7.7}$$

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 λ-**Bialg**$(\mathscr{C})$.

The pentagonal law (7.7) also has two asymmetric renderings, which relate it to liftings and coliftings.



(7.8)

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:

$$\mathsf{F}_\lambda\text{-}\mathbf{Alg}(\mathscr{C}_\mathsf{N}) \cong \lambda\text{-}\mathbf{Bialg}(\mathscr{C}) \cong (\mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C}))_{\mathsf{N}^\lambda} \ . \tag{7.9}$$

The alternative interpretations are useful to determine initial and final objects in λ-**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.

7.5 Recursion Schemes from Comonads are Adjoint Folds

We now return to the proof of Theorem 7.2 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 (Hinze, 2013). Of course, the derivation is not strictly necessary, but it helps to relate the present development to prior work (Uustalu *et al.*, 2001), 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 $f : A \to B$ 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 (7.6). 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 $(a, A) \to (b_\sharp, \mathsf{N}\, B)$.

$$\begin{array}{ccc} \mathsf{F}\, A & \xrightarrow{\mathsf{F}\, h} & \mathsf{F}\, (\mathsf{N}\, B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ A & \xrightarrow[f]{} & B \end{array} \quad \Longleftrightarrow \quad \begin{array}{ccc} \mathsf{F}\, A & \xrightarrow{\mathsf{F}\, h} & \mathsf{F}\, (\mathsf{N}\, B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b_\sharp} \\ A & \xrightarrow[h]{} & \mathsf{N}\, B \end{array} \tag{7.10}$$

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 (7.7).

$$\begin{array}{ll} & f \cdot a = b \cdot \mathsf{F}\, h \\ \Longrightarrow & \quad \{\ \mathsf{N} \text{ functor}\ \} \\ & \mathsf{N}\, f \cdot \mathsf{N}\, a = \mathsf{N}\, b \cdot \mathsf{N}\, (\mathsf{F}\, h) \\ \Longrightarrow & \quad \{\ \text{Leibniz}\ \} \\ & \mathsf{N}\, f \cdot \mathsf{N}\, a \cdot \mathsf{F}_\lambda\, c = \mathsf{N}\, b \cdot \mathsf{N}\, (\mathsf{F}\, h) \cdot \mathsf{F}_\lambda\, c \\ \Longleftrightarrow & \quad \{\ a : \mathsf{F}_\lambda\, (A, c) \to (A, c) \text{ is an } \mathsf{N}\text{-coalgebra homomorphism (7.8)}\ \} \\ & \mathsf{N}\, f \cdot c \cdot a = \mathsf{N}\, b \cdot \mathsf{N}\, (\mathsf{F}\, h) \cdot \mathsf{F}_\lambda\, c \\ \Longleftrightarrow & \quad \{\ \mathsf{N}\, f \cdot c = h\ \} \\ & h \cdot a = \mathsf{N}\, b \cdot \mathsf{N}\, (\mathsf{F}\, h) \cdot \mathsf{F}_\lambda\, c \\ \Longleftrightarrow & \quad \{\ \mathsf{F}\, h = \mathsf{F}_\lambda\, h : \mathsf{F}_\lambda\, (A, c) \to \mathsf{F}_\lambda\, (\mathsf{N}\, B, \delta\, B) \text{ is an } \mathsf{N}\text{-coalgebra homomorphism (7.8)}\ \} \\ & h \cdot a = \mathsf{N}\, b \cdot \mathsf{F}_\lambda\, (\delta\, B) \cdot \mathsf{F}\, h \end{array}$$

The proof makes essential use of the pentagonal law (7.7), the fact that a and h are N-coalgebra homomorphisms, and that F_λ preserves N-coalgebra homomorphisms. Along the way, we have derived a formula for $b_\sharp$:

$$b_\sharp = \mathsf{N}\, b \cdot \mathsf{F}_\lambda\, (\delta\, B) = \mathsf{N}\, b \cdot \lambda\, (\mathsf{N}\, B) \cdot \mathsf{F}\, (\delta\, B)\ . \tag{7.11}$$

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 (7.6) 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.

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

We have discovered an important fact: b and $b_\sharp$ are also related by the Eilenberg-Moore adjunction $\mathsf{U_N} \dashv \mathsf{Cofree_N}$ since $b_\sharp = \lfloor b \rfloor$! Using the notation for adjuncts, the right-hand side of (7.10) reads $\lfloor f \rfloor \cdot a = \lfloor b \rfloor \cdot \mathsf{F}\ \lfloor f \rfloor$. This looks suspiciously like the right-hand side of (5.8), 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 following bijection, which is indeed an instance of (5.8).

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

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 (7.10). The somewhat pedantic diagrams above explicate all the information that is implicit. 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 (7.3) is $f = \lceil\!(\lfloor b \rfloor)\!\rceil$ or, expressed using the vocabulary of (Uustalu *et al.*, 2001), $f = \epsilon\ B \cdot (\!|b_\sharp|\!)$. We record

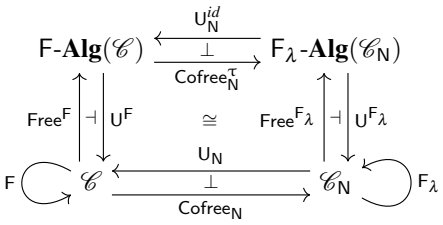
Theorem 7.4

Let $\mathsf{N} : \mathscr{C} \to \mathscr{C}$ and $\mathsf{F} : \mathscr{C} \to \mathscr{C}$ be endofunctors. A recursion scheme from the comonad N and the distributive law $\lambda : \mathsf{F} \circ \mathsf{N} \stackrel{\cdot}{\to} \mathsf{N} \circ \mathsf{F}$ can be framed as a canonical adjoint fold based on the Eilenberg-Moore adjunction $\mathsf{U_N} \dashv \mathsf{Cofree_N}$. The data functor for the adjoint fold is $\mathsf{F}_\lambda : \mathscr{C}_\mathsf{N} \to \mathscr{C}_\mathsf{N}$, the canonical control functor is $\mathsf{U_N} \circ \mathsf{F}_\lambda \circ \mathsf{Cofree_N} = \mathsf{F} \circ \mathsf{N}$, and the algebra remains $b : \mathsf{F}\ (\mathsf{N}\ B) \to B$.

$$\mathsf{F} \circ \mathsf{N} \circlearrowleft \quad \mathscr{C} \underset{\mathsf{Cofree_N}}{\overset{\mathsf{U_N}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C}_\mathsf{N} \quad \circlearrowleft \mathsf{F}_\lambda$$

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 5.6 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 (7.2b) shows that λ is an

N-coalgebra homomorphism of type $\mathsf{F}_\lambda \circ \mathsf{Cofree}_\mathsf{N} \dot{\to} \mathsf{Cofree}_\mathsf{N} \circ \mathsf{F}$.)



In the upper right corner we find the category of λ-bialgebras (7.9). 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 $\mathsf{Cofree}^\tau_\mathsf{N}\ (\mathsf{F}\ 1 \dashrightarrow 1, 1)$ is the final bialgebra.

8 When is an Adjoint Fold an Rsfc?

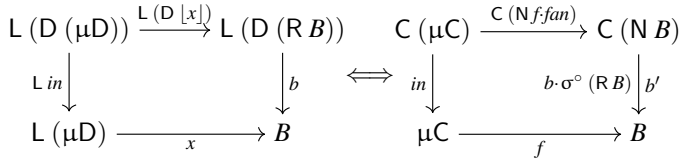
We have seen that all rsfcs are adjoint folds; and indeed, previous work has shown that the two share some connection—zygomorphisms have been modelled both by using rsfcs (Uustalu *et al.*, 2001) and as adjoint folds (Hinze, 2013). But what about the reverse direction: when can an adjoint fold also be modelled by an rsfc? In this section we show a sufficient condition: 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 a comonad (Climent & Soliveres, 2010). Consequently, we assume the following data (we rename F to C to bring the subsequent development in line with Section 5.6):

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

These are the same assumptions as for the type fusion rule, Corollary 5.6. 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 the canonical adjoint fold (5.8); on the right we have the diagram for recursion schemes from comonads (7.3).



The functions x and f are related by the isomorphism $\rho : \mathsf{L}\ (\mu\mathsf{D}) \cong \mu\mathsf{C}$, provided by Corollary 5.6. The main task is to relate the upper arrows. To this end we will use the assumption that $\mathsf{N} = \mathsf{L} \circ \mathsf{R}$ is a composition of adjoint functors to derive a simple formula

for $fan : \mu\mathsf{C} \to \mathsf{N}\ (\mu\mathsf{C})$ by working instead with $fan' : \mathsf{L}\ (\mu\mathsf{D}) \to \mathsf{N}\ (\mathsf{L}\ (\mu\mathsf{D}))$, related by $fan = \mathsf{N}\ \rho \cdot fan' \cdot \rho^\circ$.

But first, we have to set up the infrastructure. From the data above we can generate two distributive laws (Climent & Soliveres, 2010):

$$\alpha = \tau - \sigma^\circ = \mathsf{R} \circ \sigma^\circ \cdot \tau \circ \mathsf{L} : \mathsf{D} \circ \mathsf{M} \dot{\to} \mathsf{M} \circ \mathsf{D} \ , \tag{8.1a}$$

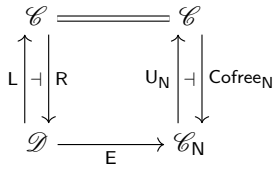
$$\gamma = \sigma^\circ - \tau = \mathsf{L} \circ \tau \cdot \sigma^\circ \circ \mathsf{R} : \mathsf{C} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{C} \ . \tag{8.1b}$$

The distributive law γ satisfies the two requirements for λ (7.2) (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:

$$\alpha \cdot \mathsf{D} \circ \eta = \eta \circ \mathsf{D} \ , \tag{8.2a}$$

$$\alpha \cdot \mathsf{D} \circ \mu = \mu \circ \mathsf{D} \cdot \mathsf{M} \circ \alpha \cdot \alpha \circ \mathsf{M} \ . \tag{8.2b}$$

Next we construct an initial γ-bialgebra. Since $\mathsf{L}^{\sigma^\circ} : \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D}) \to \mathsf{C}\text{-}\mathbf{Alg}(\mathscr{C})$ is left adjoint (5.13), and left adjoints preserve initial objects (3.8a), we know that $\mathsf{L}^{\sigma^\circ}\ (in, \mu\mathsf{D}) = (\mathsf{L}^{\sigma^\circ}\ in, \mathsf{L}\ (\mu\mathsf{D}))$ is initial in $\mathsf{C}\text{-}\mathbf{Alg}(\mathscr{C})$. To determine a formula for fan', the corresponding coalgebra part, we have to delve a bit deeper into the theory. The Eilenberg-Moore adjunction $\mathsf{U}_\mathsf{N} \dashv \mathsf{Cofree}_\mathsf{N}$ 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 $\mathsf{U}_\mathsf{N} \dashv \mathsf{Cofree}_\mathsf{N}$, depicted in the diagram below. (Since the distributive laws of the adjoint square are identities, it is actually a so-called *map of adjunctions*.)



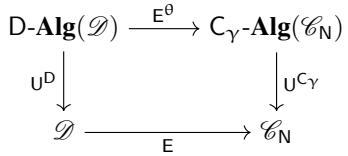
The so-called *comparison functor* $\mathsf{E} : \mathscr{D} \to \mathscr{C}_\mathsf{N}$ is defined

$$\mathsf{E}\ A = (\mathsf{L}\ A, \mathsf{L}\ (\eta\ A)) \ , \tag{8.3a}$$

$$\mathsf{E}\ f = \mathsf{L}\ f \ . \tag{8.3b}$$

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

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, where $(a, A) : \mathsf{D}\text{-}\mathbf{Alg}(\mathscr{D})$ is an arbitrary D-algebra, by lifting the comparison functor E to categories of algebras with a distributive law, as in the diagram below.



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

$$
\begin{aligned}
& \mathsf{L} \circ \eta \circ \mathsf{D} \cdot \sigma^\circ \\
= & \quad \{\ \alpha \text{ respects } \eta \text{ (8.2a)}\ \} \\
& \mathsf{L} \circ \alpha \cdot \mathsf{L} \circ \mathsf{D} \circ \eta \cdot \sigma^\circ \\
= & \quad \{\ \sigma^\circ \text{ is natural}\ \} \\
& \mathsf{L} \circ \alpha \cdot \sigma^\circ \circ \mathsf{M} \cdot \mathsf{C} \circ \mathsf{L} \circ \eta \\
= & \quad \{\ \sigma^\circ - \alpha = \sigma^\circ - \tau - \sigma^\circ = \gamma - \sigma^\circ\ \} \\
& \mathsf{N} \circ \sigma^\circ \cdot \gamma \circ \mathsf{L} \cdot \mathsf{C} \circ \mathsf{L} \circ \eta \ .
\end{aligned}
$$

We have established that $\mathit{fan}' = \mathsf{L}\,(\eta\,(\mu\mathsf{D}))$, and so an attractive definition of *fan* emerges:

$$\mathit{fan} = \mathsf{N}\,\rho \cdot \mathsf{L}\,(\eta\,(\mu\mathsf{D})) \cdot \rho^\circ \ , \tag{8.4}$$

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

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.

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

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

$$
\begin{aligned}
& \mathsf{L}\,\lfloor x \rfloor \\
= & \quad \{\ \lfloor - \rfloor \text{ expressed in terms of } \eta\text{; (3.4a) with } k,f,h := x, id, id\ \} \\
& \mathsf{L}\,(\mathsf{R}\,x \cdot \eta\,(\mu\mathsf{D})) \\
= & \quad \{\ \mathsf{L} \text{ functor}\ \} \\
& \mathsf{L}\,(\mathsf{R}\,x) \cdot \mathsf{L}\,(\eta\,(\mu\mathsf{D})) \\
= & \quad \{\ \text{definition of } \mathsf{N} \text{ and } \mathit{fan}' \text{ (8.4)}\ \} \\
& \mathsf{N}\,x \cdot \mathit{fan}'
\end{aligned}
$$

We record the result in the following

Theorem 8.1

Let $x : \mathsf{L}\,(\mu\mathsf{D}) \to B$ be a canonical adjoint fold based on the adjunction $\mathsf{L} \dashv \mathsf{R} : \mathscr{C} \rightharpoonup \mathscr{D}$ with conjugates $\sigma \dashv \tau$ where σ is an isomorphism, a data functor $\mathsf{D} : \mathscr{D} \to \mathscr{D}$, a control functor $\mathsf{C} : \mathscr{C} \to \mathscr{C}$, and an algebra $b : \mathsf{C}\,B \to B$. Since σ is an isomorphism we have $\rho : \mathsf{L}\,(\mu\mathsf{D}) \cong \mu\mathsf{C}$. The adjoint fold x can be framed in terms of a recursion scheme $f : \mu\mathsf{C} \to B$ for the comonad $\mathsf{N} = \mathsf{L} \circ \mathsf{R}$, where $x = f \cdot \rho$. The distributive law for the rsfc is $\lambda = \sigma^\circ - \tau : \mathsf{C} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{C}$, and its algebra is $b \cdot \sigma^\circ\,(\mathsf{R}\,B) : \mathsf{C}\,(\mathsf{N}\,B) \to B$.

Example 8.2
Mutumorphisms are an instance of rsfcs using $\sigma = id : \Delta \circ \mathsf{D} = \mathsf{D}^2 \circ \Delta$, see Example 5.7. Since the isomorphism is an identity, we can transform the diagram for adjoints folds almost directly into a corresponding diagram for rsfcs ($A := \mu\mathsf{D}$).

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

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 using the Eilenberg-Moore adjunction $\mathsf{U_N} \dashv \mathsf{Cofree_N}$. 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 $id : \mathsf{U_N} \circ \mathsf{F}_\lambda = \mathsf{F} \circ \mathsf{U_N}$? Recall that $\mathsf{U_N} \circ \tau = \lambda$; consequently,

$$\gamma \;=\; \sigma^\circ - \tau \;=\; \mathsf{U_N} \circ \tau \;\cdot\; id^\circ \circ \mathsf{Cofree_N} \;=\; \lambda \quad .$$

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:

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

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

9 Monadic Catamorphisms

So far we have devoted our attention to the (co)Eilenberg-Moore adjunction, since its construction on comonads has allowed us to relate adjoint folds to rsfcs. A connected line of work is to investigate the Kleisli adjunction. Its construction on monads allows us to relate adjoint folds to yet another recursion scheme: monadic catamorphisms (Fokkinga, 1994). This scheme allows monadic computations to be threaded through the traversal of a recursive structure. But first we review the construction of the Kleisli adjunction.

9.1 Background: Kleisli Categories

The Kleisli category Given a category $\mathscr{C}$ and a monad $\mathsf{M} : \mathscr{C} \to \mathscr{C}$, we can construct the Kleisli category, written $\mathscr{C}_\mathsf{M}$. This category imposes further structure on $\mathscr{C}$ by incorporating a monadic value to the target of every arrow. The objects in $\mathscr{C}_\mathsf{M}$ are the same as those in $\mathscr{C}$, but the arrows differ. A Kleisli arrow $A \to B : \mathscr{C}_\mathsf{M}$ is a $\mathscr{C}$-arrow of type $A \to \mathsf{M}\,B : \mathscr{C}$. The identity arrow of an object $X : \mathscr{C}_\mathsf{M}$ is the component $\eta\,X : X \to \mathsf{M}\,X$ of the unit of the monad. The composition of arrows $f : A \to \mathsf{M}\,B : \mathscr{C}$ and $g : B \to \mathsf{M}\,C : \mathscr{C}$ in the Kleisli category is given by $g \lhd f : A \to \mathsf{M}\,C : \mathscr{C}$, where $g \lhd f = \mu\,C \cdot \mathsf{M}\,g \cdot f$.

The Kleisli adjunction It is easy to show that the categories $\mathscr{C}$ and $\mathscr{C}_{\mathsf{M}}$ are related by an adjoint pair of functors L_{M} and R_{M}:

$$\mathscr{C}_{\mathsf{M}} \underset{\mathsf{R}_{\mathsf{M}}}{\overset{\mathsf{L}_{\mathsf{M}}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \quad .$$

The functor L_{M} promotes arrows of the base category to "pure" Kleisli arrows:

$$\mathsf{L}_{\mathsf{M}}\, A = A \ , \tag{9.1a}$$

$$\mathsf{L}_{\mathsf{M}}\, f = \eta\, B \cdot f \ , \tag{9.1b}$$

where $f : A \to B : \mathscr{C}$.

The functor R_{M} maps objects in the Kleisli category to monadic objects in the underlying category:

$$\mathsf{R}_{\mathsf{M}}\, A = \mathsf{M}\, A \ , \tag{9.2a}$$

$$\mathsf{R}_{\mathsf{M}}\, f = \mu\, B \cdot \mathsf{M}\, f \ , \tag{9.2b}$$

where the arrow $f : A \to \mathsf{M}\, B : \mathscr{C}$ is first lifted so that its source matches the expected object, and then 'rebalanced' by the multiplication $\mu : \mathsf{M} \circ \mathsf{M} \dot{\to} \mathsf{M}$ of the monad.

The unit $\eta : \mathsf{Id} \dot{\to} \mathsf{R}_{\mathsf{M}} \circ \mathsf{L}_{\mathsf{M}}$ of this adjunction is simply the unit of the monad M, and the counit $\epsilon : \mathsf{L}_{\mathsf{M}} \circ \mathsf{R}_{\mathsf{M}} \dot{\to} \mathsf{Id}$ is the identity id_{M}. A rather curious property of this adjunction is that the adjuncts merely cast arrows between the categories, where $\lfloor f \rfloor = f$ and $\lceil g \rceil = g$, as can be easily verified.

Liftings A *lifting* of the functor $\mathsf{F} : \mathscr{C} \to \mathscr{C}$ is a functor $\bar{\mathsf{F}} : \mathscr{C}_{\mathsf{M}} \to \mathscr{C}_{\mathsf{M}}$ such that

$$\mathsf{L}_{\mathsf{M}} \circ \mathsf{F} = \bar{\mathsf{F}} \circ \mathsf{L}_{\mathsf{M}} \ . \tag{9.3}$$

Recall that a distributive law $\lambda : \mathsf{F} \circ \mathsf{M} \dot{\to} \mathsf{M} \circ \mathsf{F}$ of an endofunctor F over a monad M is a natural transformation satisfying the two coherence conditions described by Equations (8.2a) and (8.2b). We can use such a distributive law to produce the lifting $\mathsf{F}_\lambda : \mathscr{C}_{\mathsf{M}} \to \mathscr{C}_{\mathsf{M}}$:

$$\mathsf{F}_\lambda\, A = \mathsf{F}\, A \ , \tag{9.4a}$$

$$\mathsf{F}_\lambda\, f = \lambda\, B \cdot \mathsf{F}\, f \ , \tag{9.4b}$$

where $f : A \to \mathsf{M}\, B : \mathscr{C}$. Beware that this notation overlaps with that of coliftings to functors over coalgebras, which we will not use in this section.

The proof that this is a lifting makes use of the distributive law:

$$\begin{aligned}
& \mathsf{L}_{\mathsf{M}}\ (\mathsf{F}\, f) \\
= \quad & \{\ \text{definition of } \mathsf{L}_{\mathsf{M}}\ (9.1\text{b})\ \} \\
& \eta\ (\mathsf{F}\, B) \cdot \mathsf{F}\, f \\
= \quad & \{\ \text{distributive law } (8.2\text{a})\ \} \\
& \lambda\, B \cdot \mathsf{F}\ (\eta\, B) \cdot \mathsf{F}\, f \\
= \quad & \{\ \mathsf{F} \text{ functor and definition of } \mathsf{F}_\lambda\ (9.4\text{b})\ \} \\
& \mathsf{F}_\lambda\ (\eta\, B \cdot f) \\
= \quad & \{\ \text{definition of } \mathsf{L}_{\mathsf{M}}\ (9.1\text{b})\ \} \\
& \mathsf{F}_\lambda\ (\mathsf{L}_{\mathsf{M}}\, f)
\end{aligned}$$

The lifting induces the conjugate $id : \mathsf{L_M} \circ \mathsf{F} \cong \mathsf{F}_\lambda \circ \mathsf{L_M}$, such that $id \dashv \tau$. Using Equation (3.12b) we can show that $\lambda = \tau \circ \mathsf{L_M}$. Since $\mathsf{L_M}$ is the identity on objects these two natural transformations have the same components.

9.2 Monadic catamorphisms are adjoint folds

A monadic catamorphism threads a computation through some recursive structure. More formally, the scheme is defined by the following equation:

Definition 9.1 (*Monadic catamorphism equation*)
Given a functor $\mathsf{F} : \mathscr{C} \to \mathscr{C}$, a monad (M, η, μ), a distributive law $\lambda : \mathsf{F} \circ \mathsf{M} \dot{\to} \mathsf{M} \circ \mathsf{F}$, and an algebra $b : \mathsf{F}\ B \to B : \mathscr{C}_\mathsf{M}$, a monadic catamorphism in the unknown $x : \mathsf{L_M}\ (in, \mu\mathsf{F}) \to \mathsf{L_M}\ (b, B)$ has the form

$$x \lhd \mathsf{L_M}\ in = b \lhd \mathsf{F}_\lambda\ x \ , \tag{9.5}$$

where the functor $\mathsf{F}_\lambda : \mathscr{C}_\mathsf{M} \to \mathscr{C}_\mathsf{M}$ is the lifting of F by λ.

The machinery we have set up earlier on in this section allows us to show how this definition corresponds to an adjoint fold. We have the following situation:

$$\mathsf{F}_\lambda \circlearrowright \mathscr{C}_\mathsf{M} \underset{\mathsf{R_M}}{\overset{\mathsf{L_M}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \circlearrowright \mathsf{F} \ ,$$

where F_λ is the lifting of F using a given distributive law $\lambda : \mathsf{F} \circ \mathsf{M} \dot{\to} \mathsf{M} \circ \mathsf{F}$.

Given an algebra $b : \mathsf{F}_\lambda\ B \to B$, we can apply our definitions to Equation (5.12) to yield:

$$\begin{array}{ccc} \mathsf{F}_\lambda\ (\mathsf{L_M}\ (\mu\mathsf{F})) & \xrightarrow{\mathsf{F}_\lambda\ x} & \mathsf{F}_\lambda\ B \\ {\scriptstyle \mathsf{L}_\mathsf{M}^{id^\circ}\ in}\downarrow & & \downarrow{\scriptstyle b} \\ \mathsf{L_M}\ (\mu\mathsf{F}) & \xrightarrow[x]{} & B \end{array} \iff \begin{array}{ccc} \mathsf{F}\ (\mu\mathsf{F}) & \xrightarrow{\mathsf{F}\ \lfloor x \rfloor} & \mathsf{F}\ (\mathsf{R_M}\ B) \\ {\scriptstyle in}\downarrow & & \downarrow{\scriptstyle \mathsf{R}_\mathsf{M}^\tau\ b} \\ \mu\mathsf{F} & \xrightarrow[\lfloor x \rfloor]{} & \mathsf{R_M}\ B \end{array}$$

The diagram on the left is in the Kleisli category, and corresponds directly to Equation (9.5). The diagram on the right is a fold $f = \lfloor x \rfloor : \mu\mathsf{F} \to \mathsf{M}\ B$ that involves only the base category $\mathscr{C}$. Some calculation gives us a clearer implementation:

$$\begin{array}{ll} & f \cdot in = \mathsf{R}_\mathsf{M}^\tau\ b \cdot \mathsf{F}\ f \\ \iff & \quad \{\ \text{definition of lifting (3.1a)}\ \} \\ & f \cdot in = \mathsf{R_M}\ b \cdot \tau\ B \cdot \mathsf{F}\ f \\ \iff & \quad \{\ \text{definition of } \mathsf{R_M}\ b \text{ (9.2b)}\ \} \\ & f \cdot in = \mu\ B \cdot \mathsf{M}\ b \cdot \tau\ B \cdot \mathsf{F}\ f \end{array}$$

This is precisely the characterization given by Fokkinga (1994).

Example 9.2
The function *accumulate* from Section 2 is a monadic fold where the underlying functor is List and the distributive law is $\lambda : \mathsf{List} \circ \mathsf{M} \dot{\to} \mathsf{M} \circ \mathsf{List}$, for an arbitrary monad M. In Haskell,

we can implement this law as:

$$\begin{aligned}&\lambda :: (\mathit{Functor}\ m, \mathit{Monad}\ m) \Rightarrow \mathsf{List}\ a\ (m\ x) \to m\ (\mathsf{List}\ a\ x)\\ &\lambda\ \mathit{Nil} \qquad\quad = \mathit{return}\ \mathit{Nil}\\ &\lambda\ (\mathit{Cons}\ a\ mx) = mx \gg\!= \lambda x \to \mathit{return}\ (\mathit{Cons}\ a\ x)\end{aligned}$$

The proof that this law respects the unit and multiplication of the monad is a simple exercise.

We are in a position to give a definition that matches the structure of a monadic fold, bearing in mind that τ and λ agree on their components:

$$\begin{aligned}&\mathit{accumulate} :: (\mathit{Functor}\ m, \mathit{Monad}\ m) \Rightarrow [m\ x] \to m\ [x]\\ &\mathit{accumulate} = (\!|\mathit{join} \cdot \mathit{fmap}\ b \cdot \lambda |\!)\\ &\quad \textbf{where}\ b\ \mathit{Nil} \qquad\quad = \mathit{return}\ [\,]\\ &\qquad\quad b\ (\mathit{Cons}\ mx\ xs) = mx \gg\!= \lambda x \to \mathit{return}\ (x : xs)\end{aligned}$$

On non-empty input, the algebra $b :: \mathsf{List}\ (m\ x)\ [x] \to m\ [x]$ appends the tail of results to the head of the list within a monadic context.

10 Calculational Properties

The calculational properties of adjoint folds were studied at depth in (Hinze, 2013). Given that rsfcs are an instance of adjoint folds, a natural question to ask is how this corresponds to the properties introduced alongside recursion schemes from comonads (Uustalu *et al.*, 2001). We start by listing the properties of adjoint folds.

Uniqueness Property The uniqueness property underpins many of the other laws, and introduces the notation $(\!|b|\!)_{\mathsf{L}}$ to represent the unique solution to the adjoint fold equation given by Theorem 5.2.

$$x = (\!|b|\!)_{\mathsf{L}} \iff x \cdot \mathsf{L}\ \mathit{in} = b \cdot \mathsf{C}\ x \cdot \sigma\ (\mu\mathsf{D})\ . \tag{10.1}$$

Computation Law An immediate consequence of the uniqueness property is the *computation law*, which simply arises by substituting the unique solution $(\!|b|\!)_{\mathsf{L}}$ into the defining equation for adjoint folds:

$$(\!|b|\!)_{\mathsf{L}} \cdot \mathsf{L}\ \mathit{in} = b \cdot \mathsf{C}\ (\!|b|\!)_{\mathsf{L}} \cdot \sigma\ (\mu\mathsf{D})\ . \tag{10.2}$$

Reflection Law The *reflection law* arises when we substitute *id* for x in the uniqueness property:

$$\mathit{id} = (\!|b|\!)_{\mathsf{L}} \iff \mathsf{L}\ \mathit{in} = b \cdot \sigma\ (\mu\mathsf{D})\ . \tag{10.3}$$

Example 10.1 (*Catamorphisms*)
We can bring these properties back to familiar territory by instantiating Equation (10.1) to the adjunction $\mathsf{Id} \dashv \mathsf{Id}$, since this gives rise to standard folds, where the canonical control functor is $\mathsf{C} = \mathsf{Id} \circ \mathsf{D} \circ \mathsf{Id} = \mathsf{D}$, and σ is simply the identity. In this setting, the uniqueness law is rendered:

$$x = (\!|b|\!) \iff x \cdot \mathit{in} = b \cdot x\ .$$

The computation law becomes the following, which is sometimes also referred to as *cancellation*:

$$(\!|b|\!) \cdot in = b \cdot \mathsf{D}\,(\!|b|\!) \tag{10.4}$$

The standard reflection law arises out of substituting the right hand side into the left:

$$id = (\!|b|\!) \iff in = b$$

Example 10.2 (*Recursion schemes from comonads*)

We have already seen that recursion schemes from comonads are an instance of adjoint folds where the adjunction is $\mathsf{U_G} \dashv \mathsf{Cofree_G}$, the data functor is F_λ, and the control functor is the canonical one. As can be expected, specializing the calculational properties above leads to the properties of rsfcs introduced by Uustalu *et al.* (2001).

For example, we can show that the computation law is in fact a generalized version of what was originally called the cancellation law. This is a simple case of plugging in definitions when the computation law for the canonical control functor is used:

$$\begin{array}{ll}
 & (\!|b|\!)_{\mathsf{U_N}} \cdot \mathsf{U_N}\ in = b \cdot \mathsf{U_N}\ (\mathsf{F}_\lambda\ (\mathsf{Cofree_N}\ (\!|b|\!)_{\mathsf{U_N}} \cdot \eta\ (\mu \mathsf{F}_\lambda))) \\
\iff & \quad \{\ \text{definition of } \mathsf{U_N}\ \} \\
 & (\!|b|\!)_{\mathsf{U_N}} \cdot in = b \cdot \mathsf{F}_\lambda\ (\mathsf{Cofree_N}\ (\!|b|\!)_{\mathsf{U_N}} \cdot \eta\ (\mu \mathsf{F}_\lambda)) \\
\iff & \quad \{\ \text{definition of } \mathsf{F}_\lambda \text{ (3.2b)}\ \} \\
 & (\!|b|\!)_{\mathsf{U_N}} \cdot in = b \cdot \mathsf{F}\ (\mathsf{Cofree_N}\ (\!|b|\!)_{\mathsf{U_N}} \cdot \eta\ (\mu \mathsf{F}_\lambda)) \\
\iff & \quad \{\ \text{definition of } \mathsf{Cofree_N} \text{ (7.5b)}\ \} \\
 & (\!|b|\!)_{\mathsf{U_N}} \cdot in = b \cdot \mathsf{F}\ (\mathsf{N}\ (\!|b|\!)_{\mathsf{U_N}} \cdot \eta\ (\mu \mathsf{F}_\lambda)) \\
\iff & \quad \{\ \text{definition of } \mathit{fan} \text{ (8.4)}\ \} \\
 & (\!|b|\!)_{\mathsf{U_N}} \cdot in = b \cdot \mathsf{F}\ (\mathsf{N}\ (\!|b|\!)_{\mathsf{U_N}} \cdot \mathit{fan})
\end{array}$$

Thus we have arrived at the comonadic recursion equation (7.3), and shown that it is solved uniquely by $(\!|b|\!)_{\mathsf{U_N}}$.

We can also reason that folding with the algebra $b = in \cdot \mathsf{F}\ (\epsilon\ (\mu \mathsf{F}))$ is the identity, which follows by showing that the right hand side of the reflection law is true.

$$\begin{array}{ll}
 & id = (\!|in \cdot (\mathsf{F} \circ \epsilon)\ (\mu \mathsf{F})|\!)_{\mathsf{U_N}} \\
\iff & \quad \{\ \text{reflection law (10.3)}\ \} \\
 & \mathsf{U_N}\ in = in \cdot (\mathsf{F} \circ \epsilon)\ (\mu \mathsf{F}) \cdot (\mathsf{U_N} \circ \mathsf{F}_\lambda \circ \eta)\ (\mu \mathsf{F}_\lambda) \\
\iff & \quad \{\ \text{definition of } \mathsf{U_N} \text{ and } \mu \mathsf{F}_\lambda = (\mu \mathsf{F}, in)\ \} \\
 & \mathsf{U_N}\ in = \mathsf{U_N}\ in \cdot (\mathsf{F} \circ \epsilon \circ \mathsf{U_N})\ (\mu \mathsf{F}_\lambda) \cdot (\mathsf{U_N} \circ \mathsf{F}_\lambda \circ \eta)\ (\mu \mathsf{F}_\lambda) \\
\iff & \quad \{\ \mathsf{F}_\lambda \text{ is a colifting}\ \} \\
 & \mathsf{U_N}\ in = \mathsf{U_N}\ in \cdot (\mathsf{F} \circ \epsilon \circ \mathsf{U_N})\ (\mu \mathsf{F}_\lambda) \cdot (\mathsf{F} \circ \mathsf{U_N} \circ \eta)\ (\mu \mathsf{F}_\lambda) \\
\iff & \quad \{\ \text{functoriality of } \mathsf{F}\ \} \\
 & \mathsf{U_N}\ in = \mathsf{U_N}\ in \cdot \mathsf{F}\ ((\epsilon \circ \mathsf{U_N})\ (\mu \mathsf{F}_\lambda) \cdot (\mathsf{U_N} \circ \eta)\ (\mu \mathsf{F}_\lambda)) \\
\iff & \quad \{\ \text{triangle identity (3.5a)}\ \} \\
 & \mathsf{U_N}\ in = \mathsf{U_N}\ in
\end{array}$$

Example 10.3 (*Catamorphisms as recursion schemes from comonads*)

Using the uniqueness property for rsfcs, we can show how to express catamorphisms as rsfcs.

$$
\begin{array}{ll}
 & (\!|b|\!)_{\mathsf{Id}} = (\!|b \cdot \mathsf{F}\ (\epsilon\ B)|\!)_{\mathsf{U_N}} \\
\iff & \quad \{\ \text{uniqueness property (10.1)}\ \} \\
 & (\!|b|\!)_{\mathsf{Id}} \cdot \mathsf{U_N}\ in = b \cdot \mathsf{F}\ (\epsilon\ B) \cdot (\mathsf{U_N} \circ \mathsf{F}_\lambda \circ \mathsf{Cofree_N})\ (\!|b|\!)_{\mathsf{Id}} \cdot (\mathsf{U_N} \circ \mathsf{F}_\lambda \circ \eta)\ (\mu \mathsf{F}_\lambda) \\
\iff & \quad \{\ \mathsf{F}_\lambda \text{ is a colifting}\ \} \\
 & (\!|b|\!)_{\mathsf{Id}} \cdot \mathsf{U_N}\ in = b \cdot \mathsf{F}\ (\epsilon\ B) \cdot (\mathsf{F} \circ \mathsf{U_N} \circ \mathsf{Cofree_N})\ (\!|b|\!)_{\mathsf{Id}} \cdot (\mathsf{F} \circ \mathsf{U_N} \circ \eta)\ (\mu \mathsf{F}_\lambda) \\
\iff & \quad \{\ \text{functoriality of } \mathsf{F}\ \} \\
 & (\!|b|\!)_{\mathsf{Id}} \cdot \mathsf{U_N}\ in = b \cdot \mathsf{F}\ (\epsilon\ B \cdot (\mathsf{U_N} \circ \mathsf{Cofree_N})\ (\!|b|\!)_{\mathsf{Id}} \cdot (\mathsf{U_N} \circ \eta)\ (\mu \mathsf{F}_\lambda)) \\
\iff & \quad \{\ \text{naturality of } \epsilon\ \} \\
 & (\!|b|\!)_{\mathsf{Id}} \cdot \mathsf{U_N}\ in = b \cdot \mathsf{F}\ ((\!|b|\!)_{\mathsf{Id}} \cdot \epsilon\ (\mu \mathsf{F}) \cdot (\mathsf{U_N} \circ \eta)\ (\mu \mathsf{F}_\lambda)) \\
\iff & \quad \{\ \text{definition of } \mathsf{U_N}\ \} \\
 & (\!|b|\!)_{\mathsf{Id}} \cdot in = b \cdot \mathsf{F}\ ((\!|b|\!)_{\mathsf{Id}} \cdot (\epsilon \circ \mathsf{U_N})\ (\mu \mathsf{F}_\lambda) \cdot (\mathsf{U_N} \circ \eta)\ (\mu \mathsf{F}_\lambda)) \\
\iff & \quad \{\ \text{triangle identity (3.5a)}\ \} \\
 & (\!|b|\!)_{\mathsf{Id}} \cdot in = b \cdot \mathsf{F}\ (\!|b|\!)_{\mathsf{Id}}
\end{array}
$$

This shows that every catamorphism is an rsfc. Or else, we can view this as a simple program optimization: an rsfc whose algebra ignores the comonadic context and merely extracts a value can be simplified to become a catamorphism.

10.1 Fusion

A useful set of laws involve the *fusion* of a fold with some arrow to produce a different fold.

Fusion Law The simplest instance of fusion, which we simply call the *fusion law*, gives the conditions required to fuse an arrow that follows after an adjoint fold.

Given an adjoint fold $(\!|a|\!)_{\mathsf{L}} : \mathscr{C}\ (\mathsf{L}\ (\mu \mathsf{D}), A)$, and an arrow $h : \mathscr{C}\ (A, B)$, we can form a new adjoint fold $(\!|b|\!)_{\mathsf{L}} : \mathscr{C}\ (\mathsf{L}\ (\mu \mathsf{D})\ B)$ under certain conditions, which we can calculate:

$$
\begin{array}{ll}
 & h \cdot (\!|a|\!)_{\mathsf{L}} = (\!|b|\!)_{\mathsf{L}} \\
\iff & \quad \{\ \text{uniqueness property (10.1)}\ \} \\
 & h \cdot (\!|a|\!)_{\mathsf{L}} \cdot \mathsf{L}\ in = b \cdot \mathsf{C}\ (h \cdot (\!|a|\!)_{\mathsf{L}} \cdot \sigma\ (\mu \mathsf{D}) \\
\iff & \quad \{\ \text{computation (10.2)}\ \} \\
 & h \cdot a \cdot \mathsf{C}\ (\!|a|\!)_{\mathsf{L}} \cdot \sigma\ (\mu \mathsf{D}) = b \cdot \mathsf{C}\ (h \cdot (\!|a|\!)_{\mathsf{L}}) \cdot \sigma\ (\mu \mathsf{D}) \\
\iff & \quad \{\ \text{functoriality of } \mathsf{C}\ \} \\
 & h \cdot a \cdot \mathsf{C}\ (\!|a|\!)_{\mathsf{L}} \cdot \sigma\ (\mu \mathsf{D}) = b \cdot \mathsf{C}\ h \cdot \mathsf{C}\ (\!|a|\!)_{\mathsf{L}} \cdot \sigma\ (\mu \mathsf{D}) \\
\Longleftarrow & \quad \{\ \text{Leibniz law}\ \} \\
 & h \cdot a = b \cdot \mathsf{C}\ h
\end{array}
$$

Summarizing, we can state the fusion law as follows:

$$
h \cdot (\!|a|\!)_{\mathsf{L}} = (\!|b|\!)_{\mathsf{L}} \quad \Longleftarrow \quad h \cdot a = b \cdot \mathsf{C}\ h\ . \tag{10.5}
$$

The arrow is required to be a C-homomorphism.

Conjugate Fusion Law The fusion law focuses on creating a new fold whose codomain is a modification of the original. Another form of fusion of interest is the so-called *conjugate*

fusion law, which focuses instead on generating a fold whose domain differs. This time, we precompose the fold $(\!|b'|\!)_{\mathsf{L}}$ by a conjugate $\alpha : \mathsf{L} \dot{\to} \mathsf{L}'$. It is instructive to derive the conditions required for this situation:

$$
\begin{array}{ll}
 & (\!|b|\!)_{\mathsf{L}'} \cdot \alpha\ (\mu\mathsf{D}) = (\!|b|\!)_{\mathsf{L}} \\
\iff & \{\ \text{uniqueness property (10.1)}\ \} \\
 & (\!|b|\!)_{\mathsf{L}'} \cdot \alpha\ (\mu\mathsf{D}) \cdot \mathsf{L}\ in = b \cdot \mathsf{C}\ ((\!|b|\!)_{\mathsf{L}'} \cdot \alpha\ (\mu\mathsf{D})) \cdot \sigma\ (\mu\mathsf{D}) \\
\iff & \{\ \text{naturality of } \alpha : \mathsf{L} \dot{\to} \mathsf{L}'\ \} \\
 & (\!|b|\!)_{\mathsf{L}'} \cdot \mathsf{L}'\ in \cdot \alpha\ (\mathsf{D}\ (\mu\mathsf{D})) = b \cdot \mathsf{C}\ ((\!|b|\!)_{\mathsf{L}'} \cdot \alpha\ (\mu\mathsf{D})) \cdot \sigma\ (\mu\mathsf{D}) \\
\iff & \{\ \text{computation law (10.2)}\ \} \\
 & b \cdot \mathsf{C}\ (\!|b|\!)_{\mathsf{L}'} \cdot \sigma'\ (\mu\mathsf{D}) \cdot \alpha\ (\mathsf{D}\ (\mu\mathsf{D})) = b \cdot \mathsf{C}\ ((\!|b|\!)_{\mathsf{L}'} \cdot \alpha\ (\mu\mathsf{D})) \cdot \sigma\ (\mu\mathsf{D}) \\
\iff & \{\ \text{functoriality of } \mathsf{C}\ \} \\
 & b \cdot \mathsf{C}\ (\!|b|\!)_{\mathsf{L}'} \cdot \sigma'\ (\mu\mathsf{D}) \cdot \alpha\ (\mathsf{D}\ (\mu\mathsf{D})) = b \cdot \mathsf{C}\ (\!|b|\!)_{\mathsf{L}'} \cdot \mathsf{C}\ (\alpha\ (\mu\mathsf{D})) \cdot \sigma\ (\mu\mathsf{D}) \\
\impliedby & \{\ \text{Leibniz law}\ \} \\
 & \sigma'\ (\mu\mathsf{D}) \cdot \alpha\ (\mathsf{D}\ (\mu\mathsf{D})) = \mathsf{C}\ (\alpha\ (\mu\mathsf{D})) \cdot \sigma\ (\mu\mathsf{D})
\end{array}
$$

We have shown that the precomposition of α with fold yields another, so long as the conjugates of the fold are related by α:

$$(\!|b|\!)_{\mathsf{L}'} \cdot \alpha\ (\mu\mathsf{D}) = (\!|b|\!)_{\mathsf{L}} \quad \impliedby \quad \sigma' \cdot \alpha \circ \mathsf{D} = \mathsf{C} \circ \alpha \cdot \sigma\ . \tag{10.6}$$

A natural transformation that satisfies the property on the right is sometimes called a transformation of conjugates.

General Fusion The previous two laws are an instance of a more general fusion law where we transform both domain and codomain at once using a natural transformation $\alpha : \mathscr{C}\ (\mathsf{L}\ -, B) \dot{\to} \mathscr{C}'\ (\mathsf{L}'\ -, B')$. We can calculate the required conditions as follows:

$$
\begin{array}{ll}
 & \alpha\ (\mu\mathsf{D})\ (\!|b|\!)_{\mathsf{L}} = (\!|b'|\!)_{\mathsf{L}'} \\
\iff & \{\ \text{uniqueness property (10.1)}\ \} \\
 & \alpha\ (\mu\mathsf{D})\ (\!|b|\!)_{\mathsf{L}} \cdot \mathsf{L}'\ in = b' \cdot \mathsf{C}'\ (\alpha\ (\mu\mathsf{D})\ (\!|b|\!)_{\mathsf{L}}) \cdot \sigma'\ (\mu\mathsf{D}) \\
\iff & \{\ \text{naturality of } \alpha\ \} \\
 & \alpha\ (\mathsf{D}\ (\mu\mathsf{D}))\ ((\!|b|\!)_{\mathsf{L}} \cdot \mathsf{L}\ in) = b' \cdot \mathsf{C}'\ (\alpha\ (\mu\mathsf{D})\ (\!|b|\!)_{\mathsf{L}}) \cdot \sigma'\ (\mu\mathsf{D}) \\
\iff & \{\ \text{computation law (10.2)}\ \} \\
 & \alpha\ (\mathsf{D}\ (\mu\mathsf{D}))\ (b \cdot \mathsf{C}\ (\!|b|\!)_{\mathsf{L}} \cdot \sigma\ (\mu\mathsf{D})) = b' \cdot \mathsf{C}'\ (\alpha\ (\mu\mathsf{D})\ (\!|b|\!)_{\mathsf{L}}) \cdot \sigma'\ (\mu\mathsf{D}) \\
\impliedby & \{\ \text{abstraction}\ \} \\
 & \alpha\ (\mathsf{D}\ X) \cdot (\lambda x\ .\ b \cdot \mathsf{C}\ x \cdot \sigma\ X) = (\lambda x\ .\ b' \cdot \mathsf{C}'\ x \cdot \sigma'\ X) \cdot \alpha\ X
\end{array}
$$

We can recover the fusion law by applying a transformation that postcomposes with $h : \mathscr{C}\ (A, B)$, which is natural in its domain: $h \cdot - : \mathscr{C}\ (\mathsf{L}\ -, A) \dot{\to} \mathscr{C}\ (\mathsf{L}\ -, B)$. Instantiating this to the general fusion law gives:

$$
\begin{array}{ll}
 & (h \cdot -)\ (\mathsf{D}\ X) \cdot (\lambda x\ .\ a \cdot \mathsf{C}\ x \cdot \sigma\ X) = (\lambda x\ .\ b \cdot \mathsf{C}\ x \cdot \sigma\ X) \cdot (h \cdot -)\ X \\
\iff & \{\ \text{extensionality}\ \} \\
 & \forall x\ .\ h \cdot a \cdot \mathsf{C}\ x \cdot \sigma\ X = b \cdot \mathsf{C}\ (h \cdot x) \cdot \sigma\ X \\
\iff & \{\ \text{functoriality of } \mathsf{C}\ \} \\
 & \forall x\ .\ h \cdot a \cdot \mathsf{C}\ x \cdot \sigma\ X = b \cdot \mathsf{C}\ h \cdot \mathsf{C}\ x \cdot \sigma\ X \\
\impliedby & \{\ \text{Leibniz}\ \} \\
 & h \cdot a = b \cdot \mathsf{C}\ h
\end{array}
$$

Similarly, the conjugate fusion law is recovered by applying a transformation that precomposes with $\alpha : \mathsf{L} \dot{\to} \mathsf{L}$, since this is a transformation of type $\mathscr{C}\ (\mathsf{L}\ -, A) \dot{\to} \mathscr{C}\ (\mathsf{L}'\ -, A)$.

11 Adjoint Unfolds

Now that we have studied adjoint folds in detail, we turn our attention to their dual counterparts: adjoint unfolds. Of course, everything dualizes as expected, but it is nevertheless instructive to see exactly how this works out in practice. We start with the definition of an adjoint unfold. To find the dual, we must remember that an adjoint fold exists only in a category that is the codomain of a left adjoint functor. Dually, an adjoint unfold exists only in a category that is the codomain of a right adjoint functor.

Definition 11.1 (*Adjoint corecursion equation*)
Given an adjunction $\mathsf{L} \dashv \mathsf{R} : \mathscr{C} \leftharpoondown \mathscr{D}$, functors $\mathsf{C} : \mathscr{C} \to \mathscr{C}$ and $\mathsf{D} : \mathscr{D} \to \mathscr{D}$, and a distributive law $\tau : \mathsf{D} \circ \mathsf{R} \dot{\to} \mathsf{R} \circ \mathsf{C}$, an adjoint corecursion equation in the unknown $y : A \to \mathsf{R}\ (\nu \mathsf{C})$ has the form

$$\mathsf{R}\ \mathit{out} \cdot y = \tau\ (\nu \mathsf{D}) \cdot \mathsf{D}\ y \cdot a \quad , \tag{11.1}$$

where $a : A \to \mathsf{D}\ A$.

Note that this definition falls out naturally from the adjoint recursion equation as a consequence of the conjugate relationship, $\sigma \dashv \tau$, between the distributive laws. It may help to view the functors C and D as the codata structure and decision structure, since their roles are now different: rather than collapsing a value of type $\mu \mathsf{D}$, we are instead constructing a value of type $\nu \mathsf{C}$, where values are decided by a D-coalgebra.

Just as before, there exists a unique solution to this equation. We can proceed with an analogous pair of steps to obtain adjoint unfolds:

First, we abstract away from the final coalgebra $(\nu \mathsf{C}, \mathit{out})$ and find the desired transposed homomorphism.

$$\begin{array}{ll} & \mathsf{R}\ b \cdot y = \tau\ B \cdot \mathsf{D}\ y \cdot a \ \ :\ \ A \to \mathsf{R}\ (\mathsf{C}\ B) \\ \iff & \{\ \lfloor - \rfloor \text{ and } \lceil - \rceil \text{ are isomorphisms (3.3)}\ \} \\ & \lceil \mathsf{R}\ b \cdot y \rceil = \lceil \tau\ B \cdot \mathsf{D}\ y \cdot a \rceil \\ \iff & \{\ \lceil - \rceil \text{ is natural (3.4b)}\ \} \\ & b \cdot \lceil y \rceil = \lceil \tau\ B \cdot \mathsf{D}\ y \rceil \cdot \mathsf{L}\ a \\ \iff & \{\ \sigma \dashv \tau \text{ conjugates (3.11b)}\ \} \\ & b \cdot \lceil y \rceil = \mathsf{C}\ \lceil y \rceil \cdot \sigma\ A \cdot \mathsf{L}\ a \\ \iff & \{\ \text{definition of colifting (3.2a)}\ \} \\ & b \cdot \lceil y \rceil = \mathsf{C}\ \lceil y \rceil \cdot \mathsf{L}_\sigma\ a \ \ :\ \ \mathsf{L}\ A \to \mathsf{C}\ B \end{array}$$

Second, we instantiate the coalgebra (B, b) to the final coalgebra $(\mu \mathsf{C}, \mathit{out})$.

Theorem 11.2 (*Adjoint unfolds*)
The adjoint corecursion equation (11.1) has the unique solution $y = \lfloor [\![\mathsf{L}_\sigma\ a]\!] \rfloor$, where $\sigma : \mathsf{L} \circ \mathsf{D} \dot{\to} \mathsf{C} \circ \mathsf{L}$ is the conjugate of τ. The arrow y is called an *adjoint unfold*.

(a)	catamorphism (Hagino, 1987; Malcolm, 1990b)	$\mathsf{Id} \dashv \mathsf{Id}$	*depth*
	mutumorphism (Fokkinga, 1990)	$\Delta \dashv (\times)$	*even/odd*
	special case: zygomorphism (Malcolm, 1990a)	$\mathsf{U}_Y \dashv \mathsf{P}_Y$	*perfect*
	special case: paramorphism (Meertens, 1992)		*wc*
	fold with a parameter (Pardo, 2002)	$- \times P \dashv (-)^P$	*cat*, *depths*
	generalised fold (Bird & Paterson, 1999)	$(- \circ \mathsf{F}) \dashv \mathsf{Ran}_\mathsf{F}$	*total*
	recursion scheme from a comonad (Uustalu *et al.*, 2001)	$\mathsf{U}_\mathsf{N} \dashv \mathsf{Cofree}_\mathsf{N}$	
	special case: histomorphism (Uustalu & Vene, 1999b)		*knapsack*
	monadic catamorphism (Fokkinga, 1994)	$\mathsf{L}_\mathsf{M} \dashv \mathsf{R}_\mathsf{M}$	*accumulate*
(b)	anamorphism (Hagino, 1987; Malcolm, 1990b)	$\mathsf{Id} \dashv \mathsf{Id}$	*from*
	(mutumorphism)	$(+) \dashv \Delta$	
	special case: apomorphism (Vene & Uustalu, 1998)		*insert*
	(generalised unfold)	$\mathsf{Lan}_\mathsf{F} \dashv (- \circ \mathsf{F})$	
	λ-coiteration (Bartels, 2003)	$\mathsf{Free}^\mathsf{M} \dashv \mathsf{U}^\mathsf{M}$	
	special case: futumorphism (Uustalu & Vene, 1999b)		

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, and $\mathsf{L}_\mathsf{M} \dashv \mathsf{R}_\mathsf{M}$ is the Kleisli adjunction.

Proof
This is an immediate consequence of finality.

$$
\begin{array}{ll}
 & \mathsf{R}\ \mathit{out} \cdot y = \tau\ (\nu \mathsf{D}) \cdot \mathsf{D}\ y \cdot a \\
\iff & \quad \{\ \text{see above}\ \} \\
 & \mathit{out} \cdot \lceil y \rceil = \mathsf{C}\ \lceil y \rceil \cdot \mathsf{L}_\sigma\ a \\
\iff & \quad \{\ (\nu \mathsf{C}, \mathit{out})\ \text{final}\ \} \\
 & \lceil y \rceil = [\![\mathsf{L}_\sigma\ a]\!] \\
\iff & \quad \{\ \lfloor - \rfloor \text{ and } \lceil - \rceil \text{ are isomorphisms (3.3)}\ \} \\
 & x = \lfloor [\![\mathsf{L}_\sigma\ a]\!] \rfloor
\end{array}
$$

□

The adjoint unfolds formulation allows us to bring all of the corecursive schemes mentioned in Section 2 into a single framework. The procedure is much the same as for the recursive counterparts we have covered: the hard part is identifying the adjunction that needs to be instantiated. We do not go through the details in this paper, but a summary of the adjoint unfolds and their corresponding instantiations are to be found in Table 1.

12 Related Work

Adjoint folds Mendler-style adjoint folds first appeared in a paper by Bird & Paterson (1999), 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 & Uustalu (2004) under the name *generalised iteration*. The first author of the present paper explored the design space of adjoint folds (Hinze, 2013), identifying the adjunctions underlying various recursion schemes. The paper also shows how to combine recursion schemes by combining the underlying adjunctions. Alas, he 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.* (2001). Simultaneously and independently, Bartels (2003) 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 (Hinze & James, 2011) 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 (1968) 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.* (1995), 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—Mac Lane (1998) 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 Climent & Soliveres (2010). Roughly speaking, they show that the Kleisli construction is a left biadjoint and the Eilenberg-Moore construction is a right biadjoint to the Huber construction.

13 Conclusion

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-**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).

An overview of the various morphisms and their duals is presented in Table 1, which shows how they are captured in the framework of adjoint (un)folds. Notice that in this paper we have restricted our attention to folds and unfolds and have not considered the more general setting of hylomorphisms. Hylomorphisms not only express the combination of a fold and an unfold, but can also be seen as a generalization of them both: folds arise when the (recursive) coalgebra in question is in° (and unfolds arise when the (corecursive) algebra is in). Very recent work by Hinze *et al.* (2015) has explored the use of recursive

coalgebras (Capretta *et al.*, 2006) and corecursive algebras in λ-bialgebras, and how this can be combined with adjoint functors and conjugate natural transformations. This has led to the introduction of *conjugate hylomorphisms*, a generalisation of hylomorphisms that covers both adjoint folds and unfolds as special cases.

14 Acknowledgements

The authors would like to thank Jeremy Gibbons for his contributions to this work. We would also like to thank the anonymous reviewers for their insight and detailed suggestions. This work has been funded by EPSRC grant number EP/J010995/1, on Unifying Theories of Generic Programming.

References

Backhouse, R., Bijsterveld, M., van Geldrop, R. and van der Woude, J. (1995) Categorical fixed point calculus. Pitt, D., Rydeheard, D. E. and Johnstone, P. (eds), *Category Theory and Computer Science: 6th International Conference, CTCS '95*. Lecture Notes in Computer Science 953, pp. 159–179. Springer.

Bartels, F. (2003) Generalised coinduction. *Mathematical Structures in Computer Science* **13**(4):321–348.

Beck, J. (1969) Distributive laws. Eckmann, B. (ed), *Seminar on Triples and Categorical Homology Theory*. Lecture Notes in Mathematics 80, pp. 119–140. Springer.

Bird, R. and de Moor, O. (1997) *Algebra of Programming*. Prentice Hall.

Bird, R. and Paterson, R. (1999) Generalised folds for nested datatypes. *Formal Aspects of Computing* **11**(2):200–222.

Capretta, V., Uustalu, T. and Vene, V. (2006) Recursive coalgebras from comonads. *Information and Computation* **204**(4):437–468.

Climent, J. and Soliveres, J. (2010) Kleisli and Eilenberg-Moore constructions as parts of biadjoint situations. *Extracta Mathematicae* **25**(1):1–61.

Eilenberg, S. and Moore, J. C. (1965) Adjoint functors and triples. *Illinois Journal of Mathematics* **9**(3):381–398.

Fokkinga, M. (1990) Tupling and mutumorphisms. *The Squiggolist* **1**(4):81–82.

Fokkinga, M. (1992) *Law and Order in Algorithmics*. PhD thesis, University of Twente.

Fokkinga, M. (1994) *Monadic Maps and Folds for Arbitrary Datatypes*. Memoranda Informatica 94-28. Department of Computer Science, University of Twente.

Gibbons, J. (2000) Generic downwards accumulations. *Science of Computer Programming* **37**(1-3):37–65.

Hagino, T. (1987) *Category Theoretic Approach to Data Types*. PhD thesis, University of Edinburgh.

Hinze, R. (2013) Adjoint folds and unfolds—an extended study. *Science of Computer Programming* **78**(11):2108–2159.

Hinze, R. and James, D. W. (2011) Proving the unique fixed-point principle correct: an adventure with category theory. *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming, ICFP '11* pp. 359–371. ACM.

Hinze, R. and Wu, N. (2011) Towards a categorical foundation for generic programming. *Proceedings of the 7th ACM SIGPLAN Workshop on Generic Programming, WGP '11* pp. 47–58. ACM.

Hinze, R. and Wu, N. (2013) Histo- and dynamorphisms revisited. *Proceedings of the 9th ACM SIGPLAN Workshop on Generic Programming, WGP '13* pp. 1–12. ACM.

Hinze, R., Wu, N. and Gibbons, J. (2013) Unifying structured recursion schemes. *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming, ICFP '13* pp. 209–220. ACM.

Hinze, R., Wu, N. and Gibbons, J. (2015) Conjugate hylomorphisms – or: The mother of all structured recursion schemes. *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '15* pp. 527–538. ACM.

Huber, P. J. (1961) Homotopy theory in general categories. *Mathematische Annalen* **144**:361–385.

Kan, D. M. (1958) Adjoint functors. *Transactions of the American Mathematical Society* **87**(2):294–329.

Kleisli, H. (1965) Every standard construction is induced by a pair of adjoint functors. *Proceedings of the American Mathematical Society* **16**(3):544–546.

Lambek, J. (1968) A fixpoint theorem for complete categories. *Mathematische Zeitschrift* **103**(2):151–161.

Mac Lane, S. (1998) *Categories for the Working Mathematician*. 2nd edn. Graduate Texts in Mathematics, vol. 5. Springer.

Malcolm, G. (1990a) *Algebraic Data Types and Program Transformation*. PhD thesis, University of Groningen.

Malcolm, G. (1990b) Data structures and program transformation. *Science of Computer Programming* **14**(2-3):255–280.

Matthes, R. and Uustalu, T. (2004) Substitution in non-wellfounded syntax with variable binding. *Theoretical Computer Science* **327**(1-2):155–174.

Meertens, L. (1992) Paramorphisms. *Formal Aspects of Computing* **4**(5):413–424.

Meijer, E., Fokkinga, M. and Paterson, R. (1991) Functional programming with bananas, lenses, envelopes and barbed wire. Hughes, J. (ed), *5th ACM Conference on Functional Programming Languages and Computer Architecture, FPCA'91*. Lecture Notes in Computer Science 523, pp. 124–144. Springer.

Mendler, N. P. (1991) Inductive types and type constraints in the second-order lambda calculus. *Annals of Pure and Applied Logic* **51**(1-2):159–172.

Palmquist, P. H. (1971) The double category of adjoint squares. Gray, J. (ed), *Reports of the Midwest Category Seminar V*. Lecture Notes in Mathematics 195, pp. 123–153. Springer.

Pardo, A. (2002) Generic accumulations. Gibbons, J. and Jeuring, J. (eds), *Generic Programming: IFIP TC2/WG2.1 Working Conference on Generic Programming*. International Federation for Information Processing 115, pp. 49–78. Kluwer Academic Publishers.

Turi, D. and Plotkin, G. (1997) Towards a mathematical operational semantics. *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, LICS '97.* pp. 280–291. IEEE Computer Society.

Uustalu, T. and Vene, V. (1999a) Mendler-style inductive types, categorically. *Nordic Journal of Computing* **6**(3):343–361.

Uustalu, T. and Vene, V. (1999b) Primitive (co)recursion and course-of-value (co)iteration, categorically. *Informatica* **10**(1):5–26.

Uustalu, T. and Vene, V. (2008) Comonadic notions of computation. *Electronic Notes in Theoretical Computer Science* **203**(5):263–284.

Uustalu, T. and Vene, V. (2011) The recursion scheme from the cofree recursive comonad. *Electronic Notes in Theoretical Computer Science* **229**(5):135–157.

Uustalu, T., Vene, V. and Pardo, A. (2001) Recursion schemes from comonads. *Nordic Journal of Computing* **8**(3):366–390.

Vene, V. and Uustalu, T. (1998) Functional programming with apomorphisms (corecursion). *Proceedings of the Estonian Academy of Sciences: Physics, Mathematics* **47**(3):147–161.