Squiggoling with Bialgebras

Recursion Schemes from Comonads Revisited

Ralf Hinze and Nicolas Wu*

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

1 Introduction

A broad class of algorithms fall within the framework of dynamic programming, where the results of smaller parts of a problem are used to build efficient solutions of the whole. Such algorithms are an essential tool in the repertoire of every skilled programmer. A classic example is the *unbounded knapsack* problem, which so neatly demonstrates how a greedy algorithm can be implemented efficiently using dynamic programming.

We start with a specification of the problem. There was once a professor who had finally reached his time to retire. His final task, before going on indefinite gardening leave, was to bring home the books and papers from his office which he treasured the most. Unfortunately, he was unable to bring them all, and could only carry a fixed weight of capacity c in his knapsack. There was no end to the number of documents he could have chosen from in his office, and so the best he could manage was to categorise them into groups, where each group i contained items whose weight and value were w_i and v_i. He was interested, of course, in computing the maximum value that would fit into his knapsack. Thus, given capacity 15 and a list of groups wvs with

$$\begin{aligned} &wvs :: [(\mathbb{N}, \mathit{Value})] \\ &wvs = [(12,4),(1,2),(2,2),(1,1),(4,10)] \ , \end{aligned}$$

the maximum value possible is 36, using three items each from the second and fifth groups. How would he proceed to efficiently choose what to take with him?

The naive recursive solution of the problem takes exponential time, since each intermediate level of the recursion spawns further recursive calls, and no common results are shared:

$$\begin{aligned} &\mathit{knapsack}_1 :: \mathbb{N} \to \mathit{Value} \\ &\mathit{knapsack}_1\, 0 = 0 \\ &\mathit{knapsack}_1\, (c+1) = \\ &\quad \mathit{maximum}'\, [v + \mathit{knapsack}_1\, (c-w) \mid (w+1, v) \leftarrow wvs, w \leqslant c] \ . \end{aligned}$$

We suppose here that the maximum value of the empty list of candidate solutions is zero, and so let $\mathit{maximum}'\, [\,] = 0$ and be the maximum of the list otherwise.

* This work has been funded by EPSRC grant number EP/J010995/1.

This example lends itself perfectly to a solution that uses dynamic programming: each recursive step can naturally be thought of as a subproblem whose result can be reused time and again. The translation into an efficient version that uses an immutable Array datatype, that is dynamically constructed to store the results, is a fairly routine exercise:

$$
\begin{array}{l}
knapsack_2 :: \mathbb{N} \to Value \\
knapsack_2\ c = table\ !\ c \\
\quad \mathbf{where} \\
\qquad table :: \mathsf{Array}\ \mathbb{N}\ Value \\
\qquad table = array\ (0, c)\ [(i, ks\ i) \mid i \leftarrow [0\mathinner{..}c]] \\
\qquad ks :: \mathbb{N} \to Value \\
\qquad ks\ i = maximum'\ [v + table\ !\ (i - w) \mid (w, v) \leftarrow wvs, w \leqslant i]\ .
\end{array}
$$

The improvement in performance is dramatic, resulting in a pseudo-polynomial time algorithm. The key to this efficiency lies in the fact that the array *table* allows constant time indexing of results that are reused in different recursive calls of the function.

However, despite the performance gains, the solution we have arrived at remains unsatisfactory. A fundamental problem with both of the implementations we have seen is that they rely on general recursion, and it has long been understood that general recursion is the 'goto' of functional programming. Of course, what we desire is a version that might be expressed as an instance of a recursion scheme that holds the promise that it terminates, and has the efficiency we expect from this algorithm.

Where do we turn to for a squiggoly answer to this problem? One approach is to use recursion schemes from comonads [1], in particular, *histomorphisms*, which are the squiggol rendering of course-of-value recursion:

Recursion schemes from comonads form a general recursion principle that makes use of a comonad $(\mathsf{N}, \epsilon, \delta)$, to provide 'contextual information' to the body of the recursion. The scheme is 'doubly generic': it is parametric in a datatype $\mu\mathsf{F}$, and in the comonad N. Histomorphisms are a particularly nice instance of the recursion scheme, where a *cofree comonad* is used to make the results of recursive calls on *all* subterms available at each iteration.

More formally, recursion schemes from comonads make use of a coalgebra $fan : \mu\mathsf{F} \to \mathsf{N}\,(\mu\mathsf{F})$ that embeds a subterm in a context. The coalgebra can be defined generically in terms of a distributive law $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$, which is subject to certain conditions (3), detailed below. Here is the scheme in its full glory:

Let $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$ be a distributive law, and $fan = (\!|\mathsf{N}\, in \cdot \lambda\,(\mu\mathsf{F})|\!)$. For any $(\mathsf{F} \circ \mathsf{N})$-algebra (B, b) there is a unique arrow $f : \mu\mathsf{F} \to B$ such that

$$
\begin{array}{ccc}
\mathsf{F}\,(\mu\mathsf{F}) & \xrightarrow{\mathsf{F}\,(\mathsf{N}\,f \cdot fan)} & \mathsf{F}\,(\mathsf{N}\,B) \\
{\scriptstyle in}\big\downarrow & & \big\downarrow{\scriptstyle b} \\
\mu\mathsf{F} & \xrightarrow[f]{} & B
\end{array} \quad . \tag{1}
$$

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

The recursion scheme is quite amazing in its generality: it works for an arbitrary functor F and an arbitrary comonad N, as long as they can be related by a distributive law. Now, the goal of this paper is to establish the correctness of the scheme, deriving the unique solution of (1) in the process. To this end we shall need quite a bit of machinery, which we shall introduce in the subsequent sections. But first let us revisit our introductory example.

To phrase the knapsack problem as an instance of the scheme we need to identify the initial algebra $(\mu\mathsf{F}, \mathit{in})$ and the comonad N. The first is easy: the type of natural numbers $\mathbb{N}$ is the initial algebra of the functor $\mathsf{Nat}\, A = 1 + A$. The second choice is specific to the class of algorithms: histomorphisms rely on the cofree comonad, in our example, on the cofree comonad for the base functor Nat, which we denote Nat_∞. We will go into more detail in the next section, but for now, Nat_∞ can be understood as the type of nonempty lists, which for our purposes, are treated as simple look-up tables. In particular, it supports a function $\mathit{lookup} :: \mathsf{Nat}_\infty\, v \to \mathbb{N} \to \mathit{Maybe}\, v$ that acts as a means of indexing values that have already been calculated.

$$
\begin{aligned}
&\mathit{knapsack}_3 :: \mathbb{N} \to \mathit{Value}\\
&\mathit{knapsack}_3 = \mathit{knap} \cdot \mathit{fmap}\,(\mathit{fmap}\,\mathit{knapsack}_3 \cdot \mathit{fan}) \cdot \mathit{in}^\circ\\
&\mathit{knap} :: \mathsf{Nat}\,(\mathsf{Nat}_\infty\; \mathit{Value}) \to \mathit{Value}\\
&\mathit{knap}\,(\mathit{Zero}) \qquad\; = 0\\
&\mathit{knap}\,(\mathit{Succ}\,\mathit{table}) =\\
&\quad \mathit{maximum}'\,[v_1 + v_2 \mid (w + 1, v_1) \leftarrow \mathit{wvs}, \mathit{Just}\; v_2 \leftarrow [\mathit{lookup}\;\mathit{table}\;w]]
\end{aligned}
$$

The helper function knap plays the part of the context-sensitive algebra. Note that $\mathit{lookup}\;\mathit{table}\;w$ corresponds to the operation $\mathit{table}\,!\,(i - w)$ of the array-based implementation. In other words, the look-up table of type $\mathsf{Nat}_\infty\; \mathit{Value}$ stores the values in reverse order.

2 Background: Comonad and Distributive Law

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

$$\epsilon \circ \mathsf{N} \cdot \delta = \mathsf{N} \;, \tag{2a}$$

$$\mathsf{N} \circ \epsilon \cdot \delta = \mathsf{N} \;, \tag{2b}$$

$$\delta \circ \mathsf{N} \cdot \delta = \mathsf{N} \circ \delta \cdot \delta \;. \tag{2c}$$

Here we use categorical notation, where natural transformations can be composed horizontally ($\circ$), and vertically ($\cdot$), and the identity natural transformation for a functor is denoted by the functor itself. 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.

In Haskell, we can capture the interface to comonads by using the following type class, where *extract* corresponds to ϵ, and *duplicate* to δ.

$$\begin{aligned}&\textbf{class}\ \mathit{Comonad}\ n\ \textbf{where}\\&\quad \mathit{extract}\ \ \ :: n\ a \to a\\&\quad \mathit{duplicate} :: n\ a \to n\ (n\ a)\ \ .\end{aligned}$$

We have already noted that histomorphisms employ the so-called *cofree comonad* of a functor F. As Haskell supports higher-kinded datatypes, the functor part of the comonad can be readily implemented as follows.

$$\begin{aligned}&\textbf{data}\ f_\infty\ a = \mathit{Cons}\ \{\mathit{head} :: a, \mathit{tail} :: f\ (f_\infty\ a)\}\\&\textbf{instance}\ (\mathit{Functor}\ f) \Rightarrow \mathit{Functor}\ (f_\infty)\ \textbf{where}\\&\quad \mathit{fmap}\ f\ (\mathit{Cons}\ a\ ts) = \mathit{Cons}\ (f\ a)\ (\mathit{fmap}\ (\mathit{fmap}\ f)\ ts)\end{aligned}$$

The type f_∞ can be seen as the type of generalised streams of observations—'generalised' because the 'tail' is a F-structure of 'streams' rather than just a single one. A generalised stream is, in fact, very similar to a generalised rose tree, except that the latter is usually seen as an element of an inductive type, whereas this construction is patently coinductive.

A cofree value can be built by coiteration from some seed value, where a given function *hd* is used to produce a value from a seed, and *tl* produces the seeds in the next level of coiteration:

$$\begin{aligned}&\mathit{coiterate} :: (\mathit{Functor}\ f) \Rightarrow (a \to b) \to (a \to f\ a) \to (a \to f_\infty\ b)\\&\mathit{coiterate}\ \mathit{hd}\ \mathit{tl}\ x = \mathit{Cons}\ (\mathit{hd}\ x)\ (\mathit{fmap}\ (\mathit{coiterate}\ \mathit{hd}\ \mathit{tl})\ (\mathit{tl}\ x))\ \ .\end{aligned}$$

The function $h = \mathit{coiterate}\ f\ c$ enjoys a universal property: it is the unique F-coalgebra homomorphism $h : (A, c) \to (\mathsf{F}_\infty\ B, \mathit{tail}\ B)$ with $f = \mathit{head}\ B \cdot h$. Together with the destructors of the datatype, *coiterate* can be used to produce an instance of the *Comonad* class:

$$\begin{aligned}&\textbf{instance}\ (\mathit{Functor}\ f) \Rightarrow \mathit{Comonad}\ (f_\infty)\ \textbf{where}\\&\quad \mathit{extract}\ \ \ = \mathit{head}\\&\quad \mathit{duplicate} = \mathit{coiterate}\ \mathit{id}\ \mathit{tail}\ \ .\end{aligned}$$

If we instantiate the base functor of the cofree comonad to Id, we obtain the type of streams. A more interesting base functor is

$$\begin{aligned}&\textbf{data}\ \mathsf{Nat}\ a = \mathit{Zero} \mid \mathit{Succ}\ a\\&\textbf{instance}\ \mathit{Functor}\ \mathsf{Nat}\ \textbf{where}\\&\quad \mathit{fmap}\ f\ \mathit{Zero}\ \ \ \ \ \ = \mathit{Zero}\\&\quad \mathit{fmap}\ f\ (\mathit{Succ}\ n) = \mathit{Succ}\ (f\ n)\ \ ,\end{aligned}$$

which gives rise to the type Nat_∞ of non-empty colists. Colists support the indexing operation that was already used in the definition of *knap*.

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

Distributive law. 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:

$$\epsilon \circ \mathsf{F} \cdot \lambda = \mathsf{F} \circ \epsilon\ , \tag{3a}$$

$$\delta \circ \mathsf{F} \cdot \lambda = \mathsf{N} \circ \lambda \cdot \lambda \circ \mathsf{N} \cdot \mathsf{F} \circ \delta\ . \tag{3b}$$

The function *coiterate* that we saw earlier can be used to create a generic distributive law for the cofree comonad. (The proof that this is, in fact, a distributive law of an endofunctor over a comonad is beyond the scope of this paper). We have $\lambda = \mathit{coiterate}\ (\mathsf{F}\ \mathit{head})\ (\mathsf{F}\ \mathit{tail})$, which is implemented as:

$$
\begin{aligned}
&\mathit{dist} :: (\mathit{Functor}\ f) \Rightarrow f\ (f_\infty\ a) \to f_\infty\ (f\ a)\\
&\mathit{dist} = \mathit{coiterate}\ (\mathit{fmap}\ \mathit{head})\ (\mathit{fmap}\ \mathit{tail})\ \ .
\end{aligned}
$$

The coalgebra *fan*, which generates the stream of all subterms, enjoys a generic definition in terms of *dist*. Below we have specialised its type to the initial algebra $\mathbb{N}$.

$$
\begin{aligned}
&\mathit{fan} :: \mathbb{N} \to \mathsf{Nat}_\infty\ \mathbb{N}\\
&\mathit{fan} = (\!|\mathit{fmap}\ \mathit{in} \cdot \mathit{dist}|\!)
\end{aligned}
$$

As an example, the call *fan* 3 generates the colist

$$\mathit{Cons}\ 3\ (\mathit{Succ}\ (\mathit{Cons}\ 2\ (\mathit{Succ}\ (\mathit{Cons}\ 1\ (\mathit{Succ}\ (\mathit{Cons}\ 0\ \mathit{Zero}))))))\ \ .$$

This corresponds to the list of all predecessors.

3 Bialgebra

The recursion scheme involves both algebras and coalgebras, and combines them in an interesting way. We have 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. In particular, *in* and *fan* satisfy the so-called

pentagonal law.

$$
\begin{array}{ccc}
\mathsf{F}\,(\mu\mathsf{F}) & \xrightarrow{\mathsf{F}\,\mathit{fan}} & \mathsf{F}\,(\mathsf{N}\,(\mu\mathsf{F})) \\
\Big\downarrow{\scriptstyle in} & & \Big\downarrow{\scriptstyle \lambda\,(\mu\mathsf{F})} \\
\mu\mathsf{F} & & \mathsf{N}\,(\mathsf{F}\,(\mu\mathsf{F})) \\
\Big\downarrow{\scriptstyle fan} & & \\
\mathsf{N}\,(\mu\mathsf{F}) & \xleftarrow{\mathsf{N}\,in} &
\end{array}
\tag{4}
$$

The diagram commutes simply because the coalgebra $\mathit{fan} = (\!|\mathsf{N}\,in \cdot \lambda\,(\mu\mathsf{F})|\!)$ is an F-homomorphism of type $(\mu\mathsf{F}, in) \to (\mathsf{N}\,(\mu\mathsf{F}), \mathsf{N}\,in \cdot \lambda\,(\mu\mathsf{F}))$, more about this shortly.

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 the distributive law $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$.

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

$$\epsilon\, C \cdot c = id_C \quad , \tag{5a}$$

$$\delta\, C \cdot c = \mathsf{N}\, c \cdot c \quad . \tag{5b}$$

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 coalgebra $(\mathsf{N}\,A, \delta\,A)$ is respectful—this is the so-called *cofree coalgebra* for the comonad N. The coherence conditions, (5a) and (5b), are just two of the comonad laws, (2a) and (2c). Coalgebras that respect ϵ and δ and N-coalgebra homomorphisms form a category, known as the *(co)-Eilenberg-Moore category* and denoted $\mathscr{C}_{\mathsf{N}}$.

The second law (5b) also enjoys an alternative reading: c is an N-coalgebra homomorphism of type $(C, c) \to (\mathsf{N}\,C, \delta\,C)$. This observation is at the heart of the Eilenberg-Moore construction, see Section 4.

Background: Bialgebra. 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* (X, a, c) consists of an F-algebra a and a coalgebra c for the comonad N such that the *pentagonal law* holds:

$$c \cdot a = \mathsf{N}\, a \cdot \lambda\, X \cdot \mathsf{F}\, c \quad . \tag{6}$$

Loosely speaking, this 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.

The pentagonal law (6) also has two asymmetric renderings

$$
\begin{array}{ccc}
\mathsf{F}\,X & \xrightarrow{\mathsf{F}\,c} & \mathsf{F}\,(\mathsf{N}\,X) \\
{\scriptstyle a}\downarrow & & \downarrow{\scriptstyle \mathsf{N}^{\lambda}\,a} \\
X & \xrightarrow[c]{} & \mathsf{N}\,X
\end{array}
\qquad
\begin{array}{ccc}
\mathsf{F}\,X & \searrow{\scriptstyle \mathsf{F}\,c} & \\
{\scriptstyle a}\downarrow & & \mathsf{F}\,(\mathsf{N}\,X) \\
X & & \downarrow{\scriptstyle \lambda\,X} \\
{\scriptstyle c}\downarrow & & \mathsf{N}\,(\mathsf{F}\,X) \\
\mathsf{N}\,X & \swarrow{\scriptstyle \mathsf{N}\,a} &
\end{array}
\qquad
\begin{array}{ccc}
X & \xleftarrow{a} & \mathsf{F}\,X \\
{\scriptstyle c}\downarrow & & \downarrow{\scriptstyle \mathsf{F}_{\lambda}\,c} \\
\mathsf{N}\,X & \xleftarrow[\mathsf{N}\,a]{} & \mathsf{N}\,(\mathsf{F}\,X)
\end{array}
\quad , \qquad (7)
$$

which relate it to so-called liftings and coliftings, which we introduce next.

Background: Lifting and colifting. A functor $\bar{\mathsf{H}} : \mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C}) \to \mathsf{G}\text{-}\mathbf{Alg}(\mathscr{D})$ is called a *lifting* of $\mathsf{H} : \mathscr{C} \to \mathscr{D}$ iff $\mathsf{H} \circ \mathsf{U}^{\mathsf{F}} = \mathsf{U}^{\mathsf{G}} \circ \bar{\mathsf{H}}$, where $\mathsf{U}^{\mathsf{F}} : \mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C}) \to \mathscr{C}$ and $\mathsf{U}^{\mathsf{G}} : \mathsf{G}\text{-}\mathbf{Alg}(\mathscr{D}) \to \mathscr{D}$ are forgetful functors.

Given a distributive law $\lambda : \mathsf{H} \circ \mathsf{F} \dot{\leftarrow} \mathsf{G} \circ \mathsf{H}$, we can define a lifting as follows:

$$
\begin{aligned}
\mathsf{H}^{\lambda}\,(A, a) &= (\mathsf{H}\,A, \mathsf{H}\,a \cdot \lambda\,A) \ , \\
\mathsf{H}^{\lambda}\,h &= \mathsf{H}\,h \ .
\end{aligned}
$$

For liftings, the action on the carrier and on homomorphisms is fixed; the action on the algebra is determined by the distributive law.

It is customary to use the action of an algebra to refer to the algebra itself. In this vein, we simplify our notation and use $\mathsf{H}^{\lambda}\,a$ to mean $\mathsf{H}^{\lambda}\,(A, a)$. We abuse this in certain contexts by using $\mathsf{H}^{\lambda}\,a$ for the arrow of the resultant algebra, $\mathsf{H}\,a \cdot \lambda\,A$.

Dually, a functor $\underline{\mathsf{H}} : \mathsf{F}\text{-}\mathbf{Coalg}(\mathscr{C}) \to \mathsf{G}\text{-}\mathbf{Coalg}(\mathscr{D})$ is called 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 a distributive law $\lambda : \mathsf{H} \circ \mathsf{F} \dot{\to} \mathsf{G} \circ \mathsf{H}$ we can define a colifting as follows:

$$
\begin{aligned}
\mathsf{H}_{\lambda}\,(C, c) &= (\mathsf{H}\,C, \lambda\,C \cdot \mathsf{H}\,c) \ , \\
\mathsf{H}_{\lambda}\,h &= \mathsf{H}\,h \ .
\end{aligned}
$$

The distributive law $\lambda : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$ underlying λ-bialgebras induces the lifting $\mathsf{N}^{\lambda} : \mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C}) \to \mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C})$. The coherence conditions (3) ensure that N^{λ} is a comonad. In particular, the natural transformations ϵ and δ are F-algebra homomorphisms of type $\epsilon\,A : \mathsf{N}^{\lambda}\,(A, a) \to (A, a)$ and $\delta\,A : \mathsf{N}^{\lambda}\,(A, a) \to \mathsf{N}^{\lambda}\,(\mathsf{N}^{\lambda}\,(A, a))$. Dually, we can use λ to colift F to the category $\mathscr{C}_{\mathsf{N}}$. Now, the coherence conditions (3) guarantee that $\mathsf{F}_{\lambda} : \mathscr{C}_{\mathsf{N}} \to \mathscr{C}_{\mathsf{N}}$ preserves respect for ϵ and δ, that is, it maps coalgebras for N to coalgebras for N.

Returning to (7), the diagram on the left shows that $c : (X, a) \to \mathsf{N}^{\lambda}\,(X, a)$ 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 (X, a, c) both as an algebra over a coalgebra $((X, c), a)$, or as a coalgebra over an algebra $((X, a), c)$.

4 Eilenberg-Moore construction

We are nearly ready to tackle the proof of uniqueness. Before we head out to the garden, we should fetch one more tool from the shed. First observe that the recursion scheme (1) involves actually two arrows: f and $\mathsf{N}\,f \cdot \mathit{fan}$. Perhaps surprisingly, the latter is an N-coalgebra homomorphism of type $(\mu\mathsf{F}, \mathit{fan}) \to (\mathsf{N}\,B, \delta\,B)$. To understand why, we delve a bit deeper into the theory.

Background: Eilenberg-Moore construction. The so-called Eilenberg-Moore construction [2] applied to comonads shows that arrows $f : A \to B$ of $\mathscr{C}$ and homomorphisms $h : (A, c) \to (\mathsf{N}\,B, \delta\,B)$ of $\mathscr{C}_{\mathsf{N}}$ are in one-to-one correspondence:

$$f = \epsilon\,B \cdot h \iff \mathsf{N}\,f \cdot c = h \ . \tag{8}$$

The homomorphism h is also called the transpose of f. To get into a squiggoly mood, let us establish the one-to-one correspondence.

"$\Longrightarrow$": We have to show that $\mathsf{N}\,f \cdot c$ is an N-coalgebra homomorphism of type $(A, c) \to (\mathsf{N}\,B, \delta\,B)$

$$\begin{aligned}
&\mathsf{N}\,(\mathsf{N}\,f \cdot c) \cdot c \\
=\;& \{\ \mathsf{N} \text{ functor and } c \text{ coalgebra for } \mathsf{N}\ \text{(5b)}\ \} \\
&\mathsf{N}\,(\mathsf{N}\,f) \cdot \delta\,A \cdot c \\
=\;& \{\ \delta \text{ natural and } f : A \to B\ \} \\
&\delta\,B \cdot \mathsf{N}\,f \cdot c\ ,
\end{aligned}$$

and that h is uniquely determined by $f = \epsilon\,B \cdot h$:

$$\begin{aligned}
&h \\
=\;& \{\ \text{comonad counit (2b)}\ \} \\
&\mathsf{N}\,(\epsilon\,B) \cdot \delta\,B \cdot h \\
=\;& \{\ h : (A, c) \to (\mathsf{N}\,B, \delta\,B)\ \} \\
&\mathsf{N}\,(\epsilon\,B) \cdot \mathsf{N}\,h \cdot c \\
=\;& \{\ \mathsf{N} \text{ functor and assumption: } f = \epsilon\,B \cdot h\ \} \\
&\mathsf{N}\,f \cdot c\ .
\end{aligned}$$

"$\Longleftarrow$": For the other direction we reason

$$\begin{aligned}
&f \\
=\;& \{\ c \text{ coalgebra for } \mathsf{N}\ \text{(5a)}\ \} \\
&f \cdot \epsilon\,A \cdot c \\
=\;& \{\ \epsilon \text{ natural and } f : A \to B\ \} \\
&\epsilon\,B \cdot \mathsf{N}\,f \cdot c \\
=\;& \{\ \text{assumption: } \mathsf{N}\,f \cdot c = h\ \} \\
&\epsilon\,B \cdot h\ .
\end{aligned}$$

5 Proof

Equipped with our shiny new tools, we can prepare the ground to solve the central problem, the proof that Equation (1) has a unique solution. Our strategy for conquering this proof is in two parts: first, we establish a bijection between certain arrows and λ-bialgebra homomorphisms; second, we instantiate the bijection to the initial λ-bialgebra. Without going into details, we assume that the ambient categories support the initial and final constructions that we will make use of.

5.1 Proof: First Half

We abstract away from the initial object $(\mu\mathsf{F}, in, fan)$, generalising to an arbitrary λ-bialgebra (A, a, c). The first 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 $h : (A, a, c) \to (\mathsf{N}\, B, b_\sharp, \delta\, B)$, where $b_\sharp$ is a to-be-determined F-algebra.

The Eilenberg-Moore construction (8) shows that arrows $f : A \to B$ and N-coalgebra homomorphisms $h : (A, c) \to (\mathsf{N}\, B, \delta\, B)$ are in one-to-one correspondence. So we identify $\mathsf{N} f \cdot c$ as the transpose of f and simplify f's equation to $f \cdot a = b \cdot \mathsf{F}\, h$.

$$\begin{array}{ccc} \mathsf{F}\,A & \xrightarrow{\mathsf{F}\,h} & \mathsf{F}(\mathsf{N}\,B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ A & \xrightarrow[f]{} & B \end{array} \iff \begin{array}{ccc} \mathsf{F}\,A & \xrightarrow{\mathsf{F}\,h} & \mathsf{F}(\mathsf{N}\,B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b_\sharp} \\ A & \xrightarrow[h]{} & \mathsf{N}\,B \\ {\scriptstyle c}\downarrow & & \downarrow{\scriptstyle \delta\,B} \\ \mathsf{N}\,A & \xrightarrow[\mathsf{N}\,h]{} & \mathsf{N}(\mathsf{N}\,B) \end{array} \tag{9}$$

"$\Longrightarrow$": We already know that $h : (A, c) \to (\mathsf{N}\, B, \delta\, B)$ is an N-coalgebra homomorphism. It remains to show that h is an F-algebra homomorphism of type $(A, a) \to (\mathsf{N}\, B, b_\sharp)$, deriving $b_\sharp$ in the calculation. The strategy for the proof is clear: we have to transmogrify f into $\mathsf{N} f \cdot c$. Thus, we apply N to both sides of $f \cdot a = b \cdot \mathsf{F}\, h$ and then 'swap' a and c using the pentagonal law (6).

$$\begin{aligned}
& f \cdot a = b \cdot \mathsf{F}\,h \\
\Longrightarrow\quad & \{\ \mathsf{N} \text{ functor }\} \\
& \mathsf{N} f \cdot \mathsf{N}\,a = \mathsf{N}\,b \cdot \mathsf{N}(\mathsf{F}\,h) \\
\Longrightarrow\quad & \{\ \text{Leibniz }\} \\
& \mathsf{N} f \cdot \mathsf{N}\,a \cdot \mathsf{F}_\lambda\,c = \mathsf{N}\,b \cdot \mathsf{N}(\mathsf{F}\,h) \cdot \mathsf{F}_\lambda\,c \\
\iff\quad & \{\ a : \mathsf{F}_\lambda(A, c) \to (A, c)\ (6)\ \} \\
& \mathsf{N} f \cdot c \cdot a = \mathsf{N}\,b \cdot \mathsf{N}(\mathsf{F}\,h) \cdot \mathsf{F}_\lambda\,c \\
\iff\quad & \{\ \mathsf{F}_\lambda\,h : \mathsf{F}_\lambda(A, c) \to \mathsf{F}_\lambda(\mathsf{N}\,B, \delta\,B) \text{ and } \mathsf{F}_\lambda\,h = \mathsf{F}\,h\ \} \\
& \mathsf{N} f \cdot c \cdot a = \mathsf{N}\,b \cdot \mathsf{F}_\lambda(\delta\,B) \cdot \mathsf{F}\,h
\end{aligned}$$

$$\begin{aligned} & \mathsf{N}\, f \cdot c \cdot a = \mathsf{N}\, b \cdot \mathsf{F}_\lambda\,(\delta\, B) \cdot \mathsf{F}\, h \\ \iff & \quad \{\ \mathsf{N}\, f \cdot c = h\ \} \\ & h \cdot a = \mathsf{N}\, b \cdot \mathsf{F}_\lambda\,(\delta\, B) \cdot \mathsf{F}\, h \end{aligned}$$

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

$$b_\sharp = \mathsf{N}\, b \cdot \mathsf{F}_\lambda\,(\delta\, B) = \mathsf{N}\, b \cdot \lambda\,(\mathsf{N}\, B) \cdot \mathsf{F}\,(\delta\, B)\ . \tag{10}$$

We have to make sure that $(\mathsf{N}\, B, b_\sharp, \delta\, B)$ is a λ-bialgebra. Since $\mathsf{F}_\lambda\,(\mathsf{N}\, B, \delta\, B)$ is a coalgebra for the comonad N, we can conclude using (8) 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, we have $b = \epsilon\, B \cdot b_\sharp$, which is essential for the reverse direction:

"$\Longleftarrow$": Again, the strategy is clear: we have to transmogrify h into $\epsilon\, B \cdot h$. Thus, we precompose both sides of the homomorphism condition with $\epsilon\, B$.

$$\begin{aligned} & h \cdot a = b_\sharp \cdot \mathsf{F}\, h \\ \implies & \quad \{\ \text{Leibniz}\ \} \\ & \epsilon\, B \cdot h \cdot a = \epsilon\, B \cdot b_\sharp \cdot \mathsf{F}\, h \\ \iff & \quad \{\ f = \epsilon\, B \cdot h \text{ and } b = \epsilon\, B \cdot b_\sharp\ \} \\ & f \cdot a = b \cdot \mathsf{F}\, h \end{aligned}$$

To summarise, f and h are related by the Eilenberg-Moore construction, as are b and $b_\sharp$.

5.2 Proof: Second Half

Now, we can reap the harvest: the initial object in the category of λ-bialgebras is $(\mu\mathsf{F}, \mathit{in}, \mathit{fan})$ where $\mathit{fan} = (\!|\mathsf{N}^\lambda\ \mathit{in}|\!) = (\!|\mathsf{N}\, \mathit{in} \cdot \lambda\,(\mu\mathsf{F})|\!)$. Several proof obligations arise. We have already noted that the pentagonal law (6) holds, see Diagram (4).

Since $(\mu\mathsf{F}, \mathit{in})$ is the initial F-algebra there is a unique F-algebra homomorphism h to any target algebra. Because of uniqueness, h is also an N-coalgebra homomorphism—recall that the coalgebra of a λ-bialgebra is simultaneously an F-algebra homomorphism.

$$\begin{array}{ccc} \mathsf{F}\,(\mu\mathsf{F}) & \xrightarrow{\mathsf{F}\, h} & \mathsf{F}\, X \\ {\scriptstyle \mathit{in}}\downarrow & & \downarrow{\scriptstyle a} \\ \mu\mathsf{F} & \underset{(\!|a|\!)}{\overset{h}{\dashrightarrow}} & X \\ {\scriptstyle \mathit{fan}}\downarrow & & \downarrow{\scriptstyle c} \\ \mathsf{N}\,(\mu\mathsf{F}) & \xrightarrow[\mathsf{N}\, h]{} & \mathsf{N}\, X \end{array}$$

It remains to show that $(\mu\mathsf{F}, \mathit{fan})$ is a coalgebra for the comonad N. The proofs make essential use of the fact that ϵ and δ are F-algebra homomorphisms. The coalgebra *fan* respects ϵ (5a):

$$
\begin{array}{ccccc}
\mathsf{F}\,(\mu\mathsf{F}) & \xleftarrow{\mathsf{F}\,(\epsilon\,(\mu\mathsf{F}))} & \mathsf{F}\,(\mathsf{N}\,(\mu\mathsf{F})) & \xleftarrow{\mathsf{F}\,\mathit{fan}} & \mathsf{F}\,(\mu\mathsf{F}) \\
{\scriptstyle \mathit{in}}\downarrow & & \downarrow{\scriptstyle \mathsf{N}^{\lambda}\,\mathit{in}} & & \downarrow{\scriptstyle \mathit{in}} \\
\mu\mathsf{F} & \xleftarrow[\epsilon\,(\mu\mathsf{F})]{} & \mathsf{N}\,(\mu\mathsf{F}) & \xleftarrow[\mathit{fan}]{} & \mu\mathsf{F}
\end{array}
\quad = \quad
\begin{array}{ccc}
\mathsf{F}\,(\mu\mathsf{F}) & \xleftarrow{\mathsf{F}\,\mathit{id}} & \mathsf{F}\,(\mu\mathsf{F}) \\
{\scriptstyle \mathit{in}}\downarrow & & \downarrow{\scriptstyle \mathit{in}} \\
\mu\mathsf{F} & \xleftarrow[\mathit{id}]{} & \mu\mathsf{F}
\end{array}
\ .
$$

It also respects δ (5b):

$$
\begin{array}{ccccc}
\mathsf{F}\,(\mathsf{N}\,(\mathsf{N}\,(\mu\mathsf{F}))) & \xleftarrow{\mathsf{F}\,(\delta\,(\mu\mathsf{F}))} & \mathsf{F}\,(\mathsf{N}\,(\mu\mathsf{F})) & \xleftarrow{\mathsf{F}\,\mathit{fan}} & \mathsf{F}\,(\mu\mathsf{F}) \\
{\scriptstyle \mathsf{N}^{\lambda}\,(\mathsf{N}^{\lambda}\,\mathit{in})}\downarrow & & \downarrow{\scriptstyle \mathsf{N}^{\lambda}\,\mathit{in}} & & \downarrow{\scriptstyle \mathit{in}} \\
\mathsf{N}\,(\mathsf{N}\,(\mu\mathsf{F})) & \xleftarrow[\delta\,(\mu\mathsf{F})]{} & \mathsf{N}\,(\mu\mathsf{F}) & \xleftarrow[\mathit{fan}]{} & \mu\mathsf{F}
\end{array}
$$

$$
= \quad
\begin{array}{ccc}
\mathsf{F}\,(\mathsf{N}\,(\mathsf{N}\,(\mu\mathsf{F}))) & \longleftarrow & \mathsf{F}\,(\mu\mathsf{F}) \\
{\scriptstyle \mathsf{N}^{\lambda}\,(\mathsf{N}^{\lambda}\,\mathit{in})}\downarrow & & \downarrow{\scriptstyle \mathit{in}} \\
\mathsf{N}\,(\mathsf{N}\,(\mu\mathsf{F})) & \longleftarrow & \mu\mathsf{F}
\end{array}
$$

$$
= \quad
\begin{array}{ccccc}
\mathsf{F}\,(\mathsf{N}\,(\mathsf{N}\,(\mu\mathsf{F}))) & \xleftarrow{\mathsf{F}\,(\mathsf{N}\,\mathit{fan})} & \mathsf{F}\,(\mathsf{N}\,(\mu\mathsf{F})) & \xleftarrow{\mathsf{F}\,\mathit{fan}} & \mathsf{F}\,(\mu\mathsf{F}) \\
{\scriptstyle \mathsf{N}^{\lambda}\,(\mathsf{N}^{\lambda}\,\mathit{in})}\downarrow & & \downarrow{\scriptstyle \mathsf{N}^{\lambda}\,\mathit{in}} & & \downarrow{\scriptstyle \mathit{in}} \\
\mathsf{N}\,(\mathsf{N}\,(\mu\mathsf{F})) & \xleftarrow[\mathsf{N}\,\mathit{fan}]{} & \mathsf{N}\,(\mu\mathsf{F}) & \xleftarrow[\mathit{fan}]{} & \mu\mathsf{F}
\end{array}
\ .
$$

Note that $\mathsf{N}\,\mathit{fan}$ is the lifting of *fan* and hence an F-homomorphism. Since there is only one homomorphism from $(\mu\mathsf{F}, \mathit{in})$ to $\mathsf{N}^{\lambda}\,(\mathsf{N}^{\lambda}\,(\mu\mathsf{F}, \mathit{in}))$, both compositions are equal.

Consequently, the unique homomorphism from the initial λ-bialgebra to the bialgebra $(\mathsf{N}\,B, b_{\sharp}, \delta\,B)$ is $h = (\!|b_{\sharp}|\!)$.

$$
\begin{array}{ccc}
\mathsf{F}\,(\mu\mathsf{F}) & \xrightarrow{\mathsf{F}\,h} & \mathsf{F}\,(\mathsf{N}\,B) \\
{\scriptstyle \mathit{in}}\downarrow & & \downarrow{\scriptstyle b_{\sharp}} \\
\mu\mathsf{F} & \xrightarrow[(\!|b_{\sharp}|\!)]{h} & \mathsf{N}\,B \\
{\scriptstyle \mathit{fan}}\downarrow & & \downarrow{\scriptstyle \delta\,B} \\
\mathsf{N}\,(\mu\mathsf{F}) & \xrightarrow[\mathsf{N}\,h]{} & \mathsf{N}\,(\mathsf{N}\,B)
\end{array}
$$

Furthermore, $f = \epsilon\, B \cdot h = \epsilon\, B \cdot (\!|b_\sharp|\!)$ is the unique solution of

$$\begin{array}{ccc} \mathsf{F}\,(\mu\mathsf{F}) & \xrightarrow{\mathsf{F}\,(\mathsf{N}\,f \cdot fan)} & \mathsf{F}\,(\mathsf{N}\,B) \\ {\scriptstyle in}\downarrow & & \downarrow{\scriptstyle b} \\ \mu\mathsf{F} & \xrightarrow[f]{} & B \end{array} \quad .$$

6 Knapsack revisited

Obtaining an efficient implementation of *knapsack* is now simply a matter of instantiating the framework above to the cofree comonad of $\mathbb{N}$.

$$\begin{array}{l} knapsack_4 :: \mathbb{N} \to Value \\ knapsack_4 = head \cdot (\!|knap_\sharp|\!) \\ (-)_\sharp :: (Functor\, f) \Rightarrow (f\,(f_\infty\, a) \to a) \to (f\,(f_\infty\, a) \to f_\infty\, a) \\ b_\sharp = fmap\; b \cdot dist \cdot fmap\; duplicate \end{array}$$

Recall that *knap* is a context-sensitive algebra of type $\mathsf{Nat}\,(\mathsf{Nat}_\infty\; Value) \to Value$: as such it has access to the recursive images of all natural numbers smaller than the current one. The implementation of $(-)_\sharp$ builds on the generic definition that works for an arbitrary comonad. As a final tweak let us simplify its implementation for the comonad at hand:

We have emphasised before that $b_\sharp$ is a coalgebra for N and consequently an N-coalgebra homomorphism. If N is the cofree comonad F_∞, then $b_\sharp$ is also an F-coalgebra homomorphism, which is the key to improving its definition. A central observation is that λ-bialgebras with $\lambda : \mathsf{F} \circ \mathsf{F}_\infty \dot{\to} \mathsf{F}_\infty \circ \mathsf{F}$ are in one-to-one correspondence to id-bialgebras with $id : \mathsf{F} \circ \mathsf{F} \dot{\to} \mathsf{F} \circ \mathsf{F}$. (The correspondence builds on the fact that the category of F-coalgebras is isomorphic to the (co)-Eilenberg-Moore category $\mathscr{C}_{\mathsf{F}_\infty}$.)

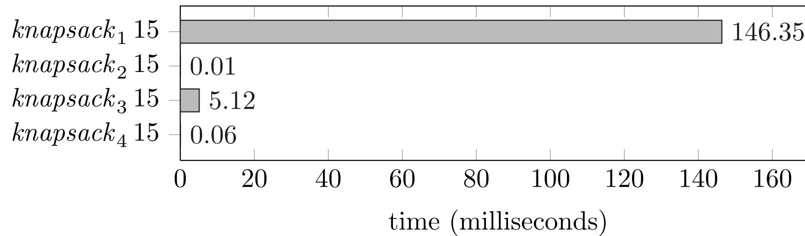
$$\begin{array}{ccc} \mathsf{F}\,X & \xrightarrow{\mathsf{F}\,c} & \mathsf{F}\,(\mathsf{F}\,X) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle id} \\ X & & \mathsf{F}\,(\mathsf{F}\,X) \\ {\scriptstyle c}\downarrow & & \\ \mathsf{F}\,X & \xleftarrow{\mathsf{F}\,a} & \end{array} \iff \begin{array}{ccc} \mathsf{F}\,X & \xrightarrow{\mathsf{F}\,\bar{c}} & \mathsf{F}\,(\mathsf{F}_\infty\,X) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle \lambda\,X} \\ X & & \mathsf{F}_\infty\,(\mathsf{F}\,X) \\ {\scriptstyle \bar{c}}\downarrow & & \\ \mathsf{F}_\infty\,X & \xleftarrow{\mathsf{F}_\infty\,a} & \end{array} \quad \textbf{where } \bar{c} = coiterate\; id\; c$$

Since the comultiplication is defined $\delta\, A = coiterate\; id\; (tail\; A)$, the id-bialgebra corresponding to the λ-bialgebra $(\mathsf{N}\,B, b_\sharp, \delta\, B)$ is $(\mathsf{N}\,B, b_\sharp, tail\; B)$. Consequently $b_\sharp$ is an F-coalgebra homomorphism: $tail\; B \cdot b_\sharp = \mathsf{F}\; b_\sharp \cdot \mathsf{F}\,(tail\; B)$. Since furthermore $head\; B \cdot b_\sharp = b$, we have $b_\sharp = coiterate\; b\; (\mathsf{F}\,(tail\; B))$.

$$\begin{array}{l} (-)_\sharp :: (Functor\, f) \Rightarrow (f\,(f_\infty\, a) \to a) \to (f\,(f_\infty\, a) \to f_\infty\, a) \\ b_\sharp = coiterate\; b\; (fmap\; tail) \end{array}$$

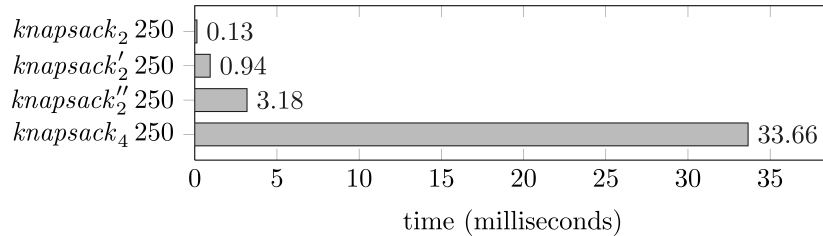
Quite interestingly, the final solution of the unbounded knapsack problem, that is, $knapsack = head \cdot (\!(coiterate\ knap\ (fmap\ tail))\!)$ is roughly a *fold of a coiteration*, a generalisation of a fold of an unfold—recall that unfolds are related to coiterations in the same way final coalgebras are related to cofree comonads.

Benchmarks. We now have four different versions of knapsack, and it is a worthwhile exercise to compare their performance with some benchmarks. The charts below show the results of the mean time from a sample of 1000 measurements. The first benchmark presents the results of solving the problem with a capacity of 15.



The results of $knapsack_1$ are underwhelming, even for very small knapsack capacities. This is entirely to be expected, given that it is an exponential algorithm, and served only as a specification of the problem.

We have claimed that our final derived version of $knapsack_4$ is efficient, so how does it compare with the array-based version discussed in Section 1? Looking more closely at $knapsack_2$ and $knapsack_4$, over a much larger capacity of 250, shows that despite our efforts, the version based on arrays is still significantly faster.



We might expect a result along these lines, given that $knapsack_2$ uses constant time look-ups in the array that is built, whereas $knapsack_4$ must still perform a linear traversal to get to its data. However, the results of $knapsack'_2$ show what happens when we replace the underlying array structure of $knapsack_2$ with a list that is treated as an indexed structure using the (!!) operator. Similarly $knapsack''_2$ is a version where the list is treated as an association list and indexed using *lookup* from the prelude. This difference in performance is rather disappointing, but note, however, that $knapsack_1$ and $knapsack_3$ were unable to complete within a reasonable time. Why is $knapsack_4$ so much slower?

The main problem occurs not in the look-up of values, but rather, in the construction of the look-up table. For $knapsack_2$, a single iteration is required

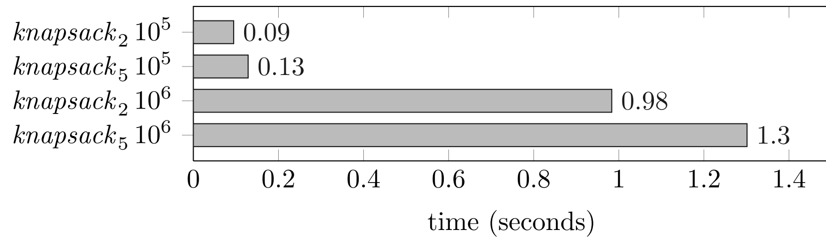
to build the table. Looking more carefully at the code that was derived for $knapsack_4$, it should be clear that the function responsible for creating the tables, *knap*, is used within a *coiterate* that is nested in a fold. Thus there is a linear factor difference between the two algorithms.

However, all is not lost, since we can use this observation to adjust our definition to become the following:

$$\begin{aligned}
&knapsack_5 :: \mathbb{N} \to Value \\
&knapsack_5 = head \cdot (\!| knap_\flat |\!) \\
&(-)_\flat :: (f\ (f_\infty\ a) \to a) \to (f\ (f_\infty\ a) \to f_\infty\ a) \\
&b_\flat\ ts = Cons\ (b\ ts)\ ts\ \ .
\end{aligned}$$

The algebra $b_\flat$ constructs the look-up table in time proportional to the running-time of b, cleverly re-using its argument for the tail of the table.

How does this compare to $knapsack_2$? The benchmarks show that the two are now in the same ball park, and their performance scales linearly. Not bad!



But what about the proof that $knapsack_5$ is correct? Does it follow the specification? What is its relationship to bialgebras? We leave the proof that $knapsack_5$ satisfies our requirements to the avid reader: without a doubt, this should be a manageable task for a distinguished professor with plenty of time on his hands.

7 Conclusion

In this paper we have given a proof of correctness of recursion schemes from comonads. Along the way, we have shown derivations of the unique arrow that solves these schemes, and presented ways of optimising this computation. Our analysis shows that the optimisations we introduced improve upon the efficiency of the standard definition of a histomorphism. Furthermore, the final version we presented, whose derivation is left as a challenge, is comparable to an array-based version. While the efficiency of our final algorithm falls slightly short of an array-based one, it gains in an important way: by construction, it is guaranteed to terminate, and we squiggolers favour correctness over speed. And so, we keenly await the derivation of our final implementation.

Acknowledgements

The authors would like to thank Jeremy Gibbons for pointing them to the knapsack problem as an interesting example of a histomorphism, and for his useful suggestions for improving this paper.

On a personal note, I would like to thank you, Doaitse, for your support and encouragement over the past fifteen years. I do hope that you enjoy your newly gained freedom. Ralf

References

1. Uustalu, T., Vene, V., Pardo, A.: Recursion schemes from comonads. Nordic J. of Computing **8** (September 2001) 366–390 2
2. Eilenberg, S., Moore, J.C.: Adjoint functors and triples. Illinois J. Math **9**(3) (1965) 381–398 8