

# Abstract Huffman Coding and PIFO Tree Embeddings

Keri D'Angelo\*      Dexter Kozen<sup>†</sup>

Cornell University  
Computer Science Department  
Ithaca, New York 14853-7501, USA

\*kd349@cornell.edu      <sup>†</sup>kozen@cs.cornell.edu

January 10, 2023

## Abstract

Algorithms for deriving Huffman codes and the recently developed algorithm for compiling PIFO trees to trees of fixed shape [1] are similar, but work with different underlying algebraic operations. In this paper, we exploit the monadic structure of prefix codes to create a generalized Huffman algorithm that has these two applications as special cases.

## 1 Introduction

*Huffman codes* translate letters from a fixed alphabet to  $d$ -ary codewords, achieving optimal compression for a given frequency distribution of letters. There is a well-known greedy algorithm for producing Huffman codes from a given distribution (see [2]).

A new data structure called a *PIFO tree* (priority-in first-out) has recently been proposed for implementing a wide range of packet scheduling algorithms in programmable network routers [3, 4]. A PIFO tree is a tree of priority queues. Currently, most routers support just a few scheduling algorithms such as strict priority or weighted fair queueing, which are baked into the hardware. The schedulers can be configured to some extent, but it is generally not possible to implement more sophisticated scheduling algorithms that require reordering of already queued packets. This is exactly what PIFO trees permit. It seems likely that PIFOs will be supported on network devices in the near future.

Some researchers have already begun to explore how the PIFO abstraction can be emulated on conventional routers [4]. In very recent work [1], it was shown how to translate an algorithm designed for a PIFO tree of arbitrary shape to one that uses a PIFO tree of fixed shape, perhaps a complete  $d$ -ary tree that might be implemented in hardware, with negligible performance degradation.

The embedding algorithm is greedy and very similar to the Huffman algorithm, except that it is based on different algebraic operations. For Huffman coding, one wishes to choose a  $d$ -ary prefix code  $C$  so as to minimize the value of  $\sum_{x \in C} |x| \cdot r(x)$ , where  $r(x)$  is the frequency of the letter assigned to the codeword  $x$ . This minimizes the entropy of the resulting code. For PIFO trees, one wishes to minimize  $\max_{x \in C} |x| + r(x)$ , where  $r(x)$  is the height of a subtree. This minimizes the height of the resulting  $d$ -ary tree and determines whether an embedding is at all possible.

This similarity leads us to seek a unified axiomatic treatment that is parametric in the algebraic operations and that can be instantiated to produce both applications as special cases. Our treatment exploits the monadic structure of prefix codes to obtain an abstract formulation of the problem and its solution. We identify sufficient conditions for our abstract algorithm to produce optimal solutions, where the meaning of *optimal* is also parametric in the instantiation.

We state axioms that are sufficient for optimality in §3. The algorithm is presented in §4 and its correctness proved in §5. The two applications of Huffman codes and PIFO trees are derived in §6.

## 2 Background

We assume familiarity with the basic category-theoretic concepts of category, functor, and natural transformation. Our exposition is based on the concepts of *monad* and *Eilenberg-Moore algebra*; we briefly review the definitions here. For a more thorough introduction, we refer the reader to [5–8].

*Monads* are heavily used in functional programming to model the augmentation of a computation with extra structure [9–11]. Formally, a *monad* on a category  $C$  is a triple  $(\mathcal{T}, \eta, \mu)$ , where  $\mathcal{T} : C \rightarrow C$  is an endofunctor on  $C$  and  $\eta : I \rightarrow \mathcal{T}$  and  $\mu : \mathcal{T}^2 \rightarrow \mathcal{T}$  are natural transformations, called the *unit* and *multiplication* respectively, such that for all objects  $X$ , the following diagrams commute:

$$\begin{array}{ccc}
 \mathcal{T}^3 X & \xrightarrow{\mu_{\mathcal{T}X}} & \mathcal{T}^2 X \\
 \mathcal{T}\mu_X \downarrow & & \downarrow \mu_X \\
 \mathcal{T}^2 X & \xrightarrow{\mu_X} & \mathcal{T}X
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{T}X & \xrightarrow{\eta_{\mathcal{T}X}} & \mathcal{T}^2 X \\
 \mathcal{T}\eta_X \downarrow & \searrow \text{id}_{\mathcal{T}X} & \downarrow \mu_X \\
 \mathcal{T}^2 X & \xrightarrow{\mu_X} & \mathcal{T}X
 \end{array}$$

Typical examples of monads are

- the *list monad*, in which  $\eta_X(a) = [a]$ , the singleton list containing  $a$ , and

$$\mu_X([[a_{11}, \dots, a_{1k_1}], \dots, [a_{n1}, \dots, a_{nk_n}]]) = [a_{11}, \dots, a_{1k_1}, \dots, a_{n1}, \dots, a_{nk_n}],$$

the list flattening operation;

- the *powerset monad*, in which  $\eta_X(a) = \{a\}$ , the singleton set containing  $a$ , and  $\mu_X(\mathcal{A}) = \bigcup \mathcal{A}$ , the operation that takes a set of subsets of  $X$  to its union.

Given a monad  $(\mathcal{T}, \eta, \mu)$  on a category  $\mathcal{C}$ , an *Eilenberg-Moore algebra* for  $(\mathcal{T}, \eta, \mu)$  is a pair  $(X, \gamma)$ , where  $X$  is an object of  $\mathcal{C}$  and  $\gamma : \mathcal{T}X \rightarrow X$  is a morphism of  $\mathcal{C}$ , called the *structure map* of the algebra, such that the following diagrams commute:

$$\begin{array}{ccc}
 \mathcal{T}^2 X & \xrightarrow{\mathcal{T}\gamma} & \mathcal{T}X \\
 \mu_X \downarrow & & \downarrow \gamma \\
 \mathcal{T}X & \xrightarrow{\gamma} & X
 \end{array}
 \qquad
 \begin{array}{ccc}
 X & & \\
 \eta_X \downarrow & \searrow \text{id}_X & \\
 \mathcal{T}X & \xrightarrow{\gamma} & X
 \end{array}$$

A *morphism of Eilenberg-Moore algebras* is a morphism of  $\mathcal{C}$  that commutes with the structure maps. That is, if  $(X, \gamma)$  and  $(Y, \delta)$  are two algebras and  $h : X \rightarrow Y$  is a morphism of  $\mathcal{C}$ , then  $h$  is a morphism of algebras  $h : (X, \gamma) \rightarrow (Y, \delta)$  if the following diagram commutes:

$$\begin{array}{ccc}
 \mathcal{T}X & \xrightarrow{\mathcal{T}h} & \mathcal{T}Y \\
 \gamma \downarrow & & \downarrow \delta \\
 X & \xrightarrow{h} & Y
 \end{array}$$

The Eilenberg-Moore algebras for  $(\mathcal{T}, \eta, \mu)$  and their morphisms form the *Eilenberg-Moore category* over the monad  $\mathcal{T}$ . The Eilenberg-Moore category for the list monad is the category of monoids and monoid homomorphisms. The Eilenberg-Moore category for the powerset monad is the category of complete upper semilattices and semilattice homomorphisms.

In our application, we will focus on the monad of *d-ary prefix codes* on the category  $\text{Set}$  of sets and set functions.

### 3 Axioms

In this section, we state the axioms that are sufficient for the optimality of our generalized Huffman algorithm.

Recall that a *prefix code* over a fixed  $d$ -ary alphabet  $\Sigma$  is a set of finite-length words over  $\Sigma$  whose elements are pairwise incomparable with respect to the prefix relation. A prefix code  $C$  is *exhaustive* if every infinite  $d$ -ary string has a prefix in  $C$ . As a consequence of König's lemma, every exhaustive prefix code over a finite alphabet is finite, but not every finite prefix code is exhaustive.

Let  $\mathcal{C} : \text{Set} \rightarrow \text{Set}$  be an endofunctor in which

- $\mathcal{C}X$  is the set of pairs  $(C, r)$  such that  $C$  is a prefix code over a  $d$ -ary alphabet for some arbitrary but fixed  $d \geq 2$  and  $r : C \rightarrow X$ , and

- for  $h : X \rightarrow Y$ ,  $\mathcal{C}h : \mathcal{C}X \rightarrow \mathcal{C}Y$  with  $\mathcal{C}h(C, r) = (C, h \circ r)$ .

The functor  $\mathcal{C}$  carries a natural monad structure with unit  $\eta : I \rightarrow \mathcal{C}$  and multiplication  $\mu : \mathcal{C}^2 \rightarrow \mathcal{C}$  defined by: for  $a \in X$  and  $(C, r) \in \mathcal{C}^2 X$  with  $r(x) = (C_x, r_x)$ ,

$$\eta_X(a) = (\{\varepsilon\}, \varepsilon \mapsto a) \quad \mu_X(C, r) = (\{xy \mid x \in C, y \in C_x\}, xy \mapsto r_x(y)).$$

The map  $xy \mapsto r_x(y)$  is well defined, as the string  $xy$  can be uniquely split into  $x \in C$  and  $y \in C_x$  because  $C$  is a prefix code.

For example, consider the prefix codes  $C = \{0, 10, 110, 111\}$  and  $C_0 = C_{10} = C_{110} = C_{111} = \{00, 11\}$  over the binary alphabet  $\{0, 1\}$ . The code  $C$  is exhaustive but the others are not. Let

$$\begin{aligned} r_0(00) &= 2 & r_{10}(00) &= 4 & r_{110}(00) &= 6 & r_{111}(00) &= 8 \\ r_0(11) &= 3 & r_{10}(11) &= 5 & r_{110}(11) &= 7 & r_{111}(11) &= 9 \\ r(0) &= (C_0, r_0) & r(10) &= (C_{10}, r_{10}) & r(110) &= (C_{110}, r_{110}) & r(111) &= (C_{111}, r_{111}). \end{aligned}$$

Then  $(C_0, r_0), (C_{10}, r_{10}), (C_{110}, r_{110}), (C_{111}, r_{111}) \in \mathcal{C}\mathbb{N}$  and  $(C, r) \in \mathcal{C}^2\mathbb{N}$ , and  $\mu_{\mathbb{N}}(C, r) = (C', r') \in \mathcal{C}\mathbb{N}$ , where

$$\begin{aligned} C' &= \{000, 011, 1000, 1011, 11000, 11011, 11100, 11111\} \\ r'(000) &= 2, r'(011) = 3, r'(1000) = 4, r'(1011) = 5, \\ r'(11000) &= 6, r'(11011) = 7, r'(11100) = 8, r'(11111) = 9. \end{aligned}$$

Suppose there is a fixed Eilenberg-Moore algebra  $(W, w)$  with  $w : \mathcal{C}W \rightarrow W$ . We call the elements of  $W$  *weights* and  $(W, w)$  a *weighting*. If  $(C, r) \in \mathcal{C}W$ , then thinking of the elements of  $C$  as a tree, the map  $r : C \rightarrow W$  assigns a weight to each leaf of the tree, and the map  $w$  tells how to assign a weight to the object  $(C, r)$  based on the leaf weights  $r$ .

To define a notion of optimality, we assume that  $W$  is totally preordered by  $\leq$ ; that is,  $\leq$  is reflexive and transitive, and for all  $x, y \in W$ , either  $x \leq y$  or  $y \leq x$  (or both). Smaller values of  $W$  in the order  $\leq$  are considered better. We write  $x \equiv y$  if both  $x \leq y$  and  $y \leq x$ . Suppose further that we have a preorder on  $\mathcal{C}W$ , also denoted  $\leq$ , satisfying the following properties.

- (i) If  $f : C \rightarrow D$  is bijective and length-nondecreasing, and if  $r \leq s \circ f$  pointwise, then  $(C, r) \leq (D, s)$ . This says that longer codewords or larger leaf values cannot cause a decrease in the order  $\leq$ .
- (ii) (Exchange property) If  $r(x) \leq r(y)$ ,  $|x| \leq |y|$ , and

$$s(z) = \begin{cases} r(x), & \text{if } z = y, \\ r(y), & \text{if } z = x, \\ r(z), & \text{if } z \in C \setminus \{x, y\}, \end{cases}$$

then  $(C, s) \leq (C, r)$ . That is, it never hurts to swap a larger element deeper in the tree with a smaller element higher in the tree.

(iii) The monad structure maps  $\eta_W : W \rightarrow \mathcal{C}W$  and  $\mu_W : \mathcal{C}^2W \rightarrow \mathcal{C}W$  are monotone with respect to  $\leq$ , where  $\leq$  on  $\mathcal{C}^2W$  is defined by:

$$(C, r) \leq (D, s) \Leftrightarrow \mathcal{C}w(C, r) \leq \mathcal{C}w(D, s).$$

Some special cases of (i) are

- If  $f : C \rightarrow D$  is bijective and length-nondecreasing, then  $(C, s \circ f) \leq (D, s)$ . Thus lengthening codewords cannot cause  $\leq$  to decrease.
- If  $f : C \rightarrow D$  is bijective and length-preserving, then  $(C, s \circ f) \equiv (D, s)$ . This says that the order  $\leq$  on trees depends only on the lengths of the codewords in  $C$ , not on the actual codewords themselves.
- If  $r, s : C \rightarrow W$  and  $r \leq s$  pointwise, then  $(C, r) \leq (C, s)$ . Thus larger leaf values cannot cause  $\leq$  to decrease.

We assume these properties hold for the algorithm described in the next section.

For  $(C, r), (D, s) \in \mathcal{C}W$ , let us write  $(C, r) \sim (D, s)$  if the multisets of weights represented by the two objects are the same; that is, there is a bijective function  $f : C \rightarrow D$  such that  $r = s \circ f$ . A tree  $(C, r) \in \mathcal{C}W$  is defined to be *optimal* (for its multiset of weights) if  $(C, r)$  is  $\leq$ -minimum in its  $\sim$ -class; that is,  $(C, r) \leq (D, s)$  for all  $(D, s)$  such that  $(C, r) \sim (D, s)$ .

We will give two detailed examples in §6.

## 4 Algorithm

Suppose we are given a multiset  $M$  of weights in  $W$ ,  $|M| \geq 2$ . We would like to find an optimal tree for this multiset of weights. The following is a recursive algorithm to find such an optimal tree.

1. Say there are  $n \geq 2$  elements in  $M$ . Let  $k \in \{2, \dots, d\}$  such that  $n \equiv k \pmod{d-1}$ . Let  $a_0, \dots, a_{k-1}$  be the  $k$  elements of least weight. Form the object

$$(\{0, 1, \dots, k-1\}, i \mapsto a_i) \in \mathcal{C}W.$$

If there are no other elements of  $M$ , return that object.

2. Otherwise, let

$$M' = \{(\{0, 1, \dots, k-1\}, i \mapsto a_i)\} \cup \{\eta_W(a) \mid a \in M \setminus \{a_0, \dots, a_{k-1}\}\},$$

a multiset of  $n - k + 1 < n$  elements of  $\mathcal{C}W$ .

3. Recursively call the algorithm at step 1 with  $M'' = \{w(E, t) \mid (E, t) \in M'\}$ , a multiset of elements of  $W$ . This returns a tree  $(D, s)$  of type  $\mathcal{C}W$  that is optimal for  $M''$ . The bijective map  $s : D \rightarrow M''$  factors as  $w \circ s'$  for some bijective  $s' : D \rightarrow M'$ , and  $(D, s') \in \mathcal{C}^2W$  with  $\mathcal{C}w(D, s') = (D, w \circ s') = (D, s)$ . Flatten this to  $\mu_W(D, s') \in \mathcal{C}W$  and return that value.

Note that the number of items combined in step 1 will be  $d$  in all recursive calls except possibly the first. This is because in every step, if  $k \in \{2, 3, \dots, d\}$ , then after that step the number of remaining elements will be  $(c(d-1) + k) - k + 1 = c(d-1) + 1$ , which is congruent to  $d \pmod{d-1}$ , so  $d$  elements will be taken in the next step. But from that point on, it is an invariant of the recursion that the number of elements remaining is  $1 \pmod{d-1}$ , since in each step we remove  $d$  elements and add one back, decreasing the number by  $d-1$ .

## 5 Correctness

In this section, we prove the correctness of the algorithm, making use of the following lemma.

**Lemma 1.** *Let  $k \in \{2, 3, \dots, d\}$  and  $k \equiv |M| \pmod{d-1}$ . Let  $a_0, \dots, a_{k-1}$  be the  $k$  elements of  $M$  of least weight, listed in nondecreasing order of weight. There is an optimal tree in  $\mathcal{CW}$  in which  $a_0, \dots, a_{k-1}$  are sibling leaves at the deepest level and have no other siblings.*

*Proof.* Let  $(C, r) \in \mathcal{CW}$  be optimal. Axiom (i) allows us to transform  $(C, r)$  so that there are no deficient nodes (nodes with fewer than  $d$  children) at any level except the deepest, and only one deficient node at the deepest level. Thus we can assume without loss of generality that there are  $k$  elements  $x_0, \dots, x_{k-1} \in C$  of maximum length  $n$  in  $C$  with a common prefix of length  $n-1$ , and no other  $y \in C$  has that prefix. Say the  $x_0, \dots, x_{k-1}$  are listed in nondecreasing order of  $r(x_i)$ ; that is,  $r(x_i) \leq r(x_j)$  for all  $0 \leq i \leq j \leq k-1$ . Let  $y_0, \dots, y_{k-1} \in C$  such that  $r(y_i) = a_i$ . Since the  $a_i$  are minimal,  $r(y_i) \leq r(x_i)$ . Because the  $|x_i|$  are of maximum length,  $|y_i| \leq |x_i|$ . Now we can swap using axiom (ii). Let

$$s(z) = \begin{cases} r(x_i), & \text{if } z = y_i, \\ r(y_i), & \text{if } z = x_i, \\ r(z), & \text{otherwise.} \end{cases}$$

Then  $(C, s) \leq (C, r)$ . But since  $(C, r)$  was optimal,  $(C, r) \equiv (C, s)$  and  $(C, s)$  is also optimal.  $\square$

**Theorem 2.** *The algorithm of §4 produces an optimal tree.*

*Proof.* By induction on  $n$ . The basis is  $n \leq d$ , in which case the result is straightforward.

Suppose that we have a multiset  $M$  of  $n > d$  elements of  $W$ . Let  $(C, r)$  be an optimal tree for  $M$ . Let  $k \in \{2, 3, \dots, d\}$  be congruent mod  $d-1$  to  $|M|$ . Let  $a_0, \dots, a_{k-1}$  be the  $k$  smallest elements of  $M$ . By Lemma 1, we can assume without loss of generality that  $a_0, \dots, a_{k-1}$  are siblings and occur at maximum depth in  $(C, r)$ , so there exist strings  $x_0, x_1, \dots, x_{k-1} \in C$  of maximum length with a common prefix  $x$  and  $r(x_i) = a_i$ . Remove the strings  $x_i$  from  $C$  and replace them with  $x$ . Call the resulting set  $C'$ . For  $z \in C'$ , let

$$r'(z) = \begin{cases} (\{0, 1, \dots, k-1\}, i \mapsto a_i), & \text{if } z = x, \\ \eta_W(r(z)), & \text{otherwise.} \end{cases}$$

Then  $(C', r') \in \mathcal{C}^2W$  and  $(C, r) = \mu_W(C', r')$ . The multiset of values of  $r'$  is just the  $M'$  of step 2 of the algorithm.

The algorithm will form the multiset

$$M'' = \{w(E, t) \mid (E, t) \in M'\} = \{w(r'(z)) \mid z \in C'\}$$

and recursively call with these weights. By the induction hypothesis, the return value will be a tree  $(D, s) \in \mathcal{C}W$  that is optimal for  $M''$ , thus  $(D, s) \leq (C', w \circ r')$ , and the bijective map  $s : D \rightarrow M''$  factors as  $s = w \circ r' \circ f$  for some bijective  $f : D \rightarrow C'$ . Let  $s' = r' \circ f$ . By axiom (iii),

$$\mathcal{C}w(D, s') = (D, w \circ s') = (D, s) \leq (C', w \circ r') = \mathcal{C}w(C', r'),$$

therefore  $(D, s') \leq (C', r')$ , and since  $\mu_W$  is monotone,

$$\mu_W(D, s') \leq \mu_W(C', r') = (C, r).$$

As  $(C, r)$  was optimal, so is  $\mu_W(D, s')$ , and this is the value returned by the algorithm.  $\square$

## 6 Applications

By choosing two specific weightings  $(W, w)$  and defining the ordering relations  $\leq$  appropriately, we can recover two special cases of this algorithm.

### 6.1 Huffman coding

Our first application is Huffman codes. Here we wish to minimize the expected length of variable-length codewords, given frequencies of the letters to be coded. For this application, we take  $W = \mathbb{R}_+ = \{a \in \mathbb{R} \mid a \geq 0\}$  with weighting

$$w(C, r) = \sum_{x \in C} r(x).$$

Recall that for  $a \in W$  and  $(C, r) \in \mathcal{C}^2W$  with  $r(x) = (C_x, r_x)$ ,

$$\eta_W(a) = (\{\varepsilon\}, \varepsilon \mapsto a) \quad \mu_W(C, r) = (\{xy \mid x \in C, y \in C_x\}, xy \mapsto r_x(y)).$$

Then  $(W, w)$  is an Eilenberg-Moore algebra for the monad  $(\mathcal{C}, \mu, \eta)$ , as

$$\begin{aligned} w(\eta_W(a)) &= w(\{\varepsilon\}, \varepsilon \mapsto a) = \sum_{x \in \{\varepsilon\}} (\varepsilon \mapsto a)(x) = a, \\ w(\mu_W(C, r)) &= \sum_{x \in C} \sum_{y \in C_x} r_x(y) = \sum_{x \in C} w(C_x, r_x) \\ &= \sum_{x \in C} w(r(x)) = w(C, w \circ r) = w(\mathcal{C}w(C, r)). \end{aligned}$$

In addition, let us define  $\alpha : \mathcal{C}W \rightarrow W$  by

$$\alpha(C, r) = \sum_{x \in C} |x| \cdot r(x).$$

**Lemma 3.**

$$\alpha(\eta_W(a)) = 0 \quad \alpha(\mu_W(C, r)) = \alpha(C, w \circ r) + w(C, \alpha \circ r).$$

*Proof.*

$$\begin{aligned} \alpha(\eta_W(a)) &= \alpha(\{\varepsilon\}, \varepsilon \mapsto a) = \sum_{x \in \{\varepsilon\}} |x| \cdot (\varepsilon \mapsto a)(x) = |\varepsilon| \cdot a = 0, \\ \alpha(\mu_W(C, r)) &= \alpha(\{xy \mid x \in C, y \in C_x\}, xy \mapsto r_x(y)) \\ &= \sum_{x \in C} \sum_{y \in C_x} |xy| \cdot r_x(y) = \sum_{x \in C} |x| \sum_{y \in C_x} r_x(y) + \sum_{x \in C} \sum_{y \in C_x} |y| \cdot r_x(y) \\ &= \sum_{x \in C} |x| \cdot w(C_x, r_x) + \sum_{x \in C} \alpha(C_x, r_x) = \sum_{x \in C} |x| \cdot w(r(x)) + \sum_{x \in C} \alpha(r(x)) \\ &= \alpha(C, w \circ r) + w(C, \alpha \circ r). \quad \square \end{aligned}$$

Note that  $\alpha$  and  $w$  agree on trees of depth one:

$$\begin{aligned} w(\{0, 1, \dots, k-1\}, i \mapsto a_i) &= \sum_{i=0}^{k-1} a_i, \\ \alpha(\{0, 1, \dots, k-1\}, i \mapsto a_i) &= \sum_{i=0}^{k-1} |i| \cdot a_i = \sum_{i=0}^{k-1} a_i, \end{aligned}$$

where  $|i|$  refers to the length of  $i$  as a string, which in this case is 1.

The map  $\alpha$  is related to the Shannon entropy  $H$ . If  $r(x) = d^{-|x|}$ , the probability of a  $d$ -ary codeword  $x$  under the uniform distribution on a  $d$ -ary alphabet, then

$$H(C, r) = \sum_{x \in C} -d^{-|x|} \log d^{-|x|} = \sum_{x \in C} |x| \cdot d^{-|x|} \log d = \alpha(C, r) \log d,$$

so  $\alpha(C, r) = H(C, r) / \log d$ .

To use the algorithm in §4, we need an order  $\leq$  on  $\mathcal{CW}$ . Define  $(C, r) \leq (D, s)$  if  $(C, r) \sim (D, s)$ , that is, there is a bijective map  $f : C \rightarrow D$  such that  $r = s \circ f$ , and

$$\alpha(C, r) \leq \alpha(D, s).$$

Note that if  $(C, r) \leq (D, s)$ , then

$$w(C, r) = \sum_{x \in C} r(x) = \sum_{x \in C} s(f(x)) = \sum_{y \in D} s(y) = w(D, s).$$

According to axiom (iii), for  $(C, r), (D, s) \in \mathcal{C}^2W$ ,

$$\begin{aligned} (C, r) \leq (D, s) &\Leftrightarrow \mathcal{C}w(C, r) \leq \mathcal{C}w(D, s) \\ &\Leftrightarrow \alpha(\mathcal{C}w(C, r)) \leq \alpha(\mathcal{C}w(D, s)) \\ &\Leftrightarrow \alpha(C, w \circ r) \leq \alpha(D, w \circ s). \end{aligned} \quad (1)$$

Also, if  $(C, r) \leq (D, s)$  in  $\mathcal{C}^2W$ , then

$$w(C, \alpha \circ r) = \sum_{x \in C} \alpha(r(x)) = \sum_{x \in C} \alpha(s(f(x))) = \sum_{y \in D} \alpha(s(y)) = w(D, \alpha \circ s). \quad (2)$$

**Lemma 4.**  $\mu_W : \mathcal{C}^2W \rightarrow \mathcal{C}W$  and  $\eta_W : W \rightarrow \mathcal{C}W$  are monotone with respect to  $\leq$ .

*Proof.* For  $\eta_W$ , suppose  $a, b \in W$  and  $a \leq b$ . By Lemma 3,

$$\alpha(\eta_W(a)) = 0 = \alpha(\eta_W(b)) \quad w(\eta_W(a)) = a \leq b = w(\eta_W(b)).$$

For  $\mu_W$ , suppose  $(C, r), (D, s) \in \mathcal{C}^2W$  and  $(C, r) \leq (D, s)$ . By Lemma 3, (1), and (2),

$$\begin{aligned} \alpha(\mu_W(C, r)) &= \alpha(C, w \circ r) + w(C, \alpha \circ r) \\ &\leq \alpha(D, w \circ s) + w(D, \alpha \circ s) = \alpha(\mu_W(D, s)). \end{aligned} \quad \square$$

**Theorem 5.** The algorithm in §4 for the algebra  $(\mathbb{R}_+, w)$  and ordering relation  $\leq$  defined by  $\alpha$  is equivalent to Huffman's algorithm and produces an optimal Huffman code for a given multiset of weights.

*Proof.* Take  $X \subset \mathbb{R}_+$  to be a finite multiset and sort the set  $X$  in increasing order. For the binary case of Huffman codes (the  $d$ -ary version follows the same way), we always choose  $k = 2$ . For the first step, let  $a_0, a_1 \in X$  be the two smallest elements in the list. Form the object  $(\{0, 1\}, i \mapsto a_i) \in \mathcal{C}X$ . In the case  $n = 2$ , this is the only remaining object in the list. Otherwise, we combined them into one element with the sum of the weights of  $a_0$  and  $a_1$  as the weight of the new element, exactly as the Huffman coding does.

For the case  $n > 2$ , there are remaining elements in the set  $X$ . Take all remaining  $a \in X \setminus \{a_0, a_1\}$  and replace  $a$  by  $\eta_X(a) \in \mathcal{C}X$ . We are left with  $n - 1$  elements of type  $\mathcal{C}X$ . If we recursively call the algorithm in step 1, we are continually combining the least two elements in the remaining set with the elements weighted by  $w$ . Note by the weighting  $w$ ,  $w(\eta_X(a)) = a$  and on elements in  $\mathcal{C}X$ ,  $w$  takes the sum of  $r(x)$ 's, exactly as Huffman coding does. Finally, this leaves us with a tree in  $\mathcal{C}^2X$  where leaves have weights of the form  $\eta_X(a_i)$ . Denote this tree by  $(D, s)$ . Taking  $\mu_X(D, S)$  gives our desired tree in  $\mathcal{C}X$ .  $\square$

## 6.2 PIFO trees

PIFO trees were introduced in [3] as a model for programmable packet schedulers. In the recent work of [1], further work was done on PIFO trees giving a semantics that allows for certain embedding algorithms. The notion of a *homomorphic embedding* was defined for the purpose determining when a PIFO tree could be represented by another PIFO tree and for finding an embedding if so. The embedding algorithm we consider takes an arbitrary PIFO tree and embeds it into a  $d$ -ary tree. This becomes a special case of the algorithm of §4, where we choose  $w$  in the weighting  $(W, w)$  to minimize the height of the target  $d$ -ary tree into which the source tree can embed.

For this application, we take  $W = \mathbb{N}$  with weighting

$$w(C, r) = \max_{x \in C} |x| + r(x).$$

This gives an Eilenberg-Moore algebra  $(W, w)$  for the monad  $(\mathcal{C}, \mu, \eta)$ . For  $a \in W$  and  $(C, r) \in \mathcal{C}^2W$  with  $r(x) = (C_x, r_x)$ , as before we have

$$\eta_W(a) = (\{\varepsilon\}, \varepsilon \mapsto a) \quad \mu_W(C, r) = (\{xy \mid x \in C, y \in C_x\}, xy \mapsto r_x(y)),$$

so

$$\begin{aligned}
w(\eta_W(a)) &= w(\{\varepsilon\}, \varepsilon \mapsto a) = \max_{x \in \{\varepsilon\}} |x| + (\varepsilon \mapsto a)(x) = |\varepsilon| + a = a, \\
w(\mu_W(C, r)) &= w(\{xy \mid x \in C, y \in C_x\}, xy \mapsto r_x(y)) = \max_{x \in C} \max_{y \in C_x} |xy| + r_x(y) \\
&= \max_{x \in C} \max_{y \in C_x} |x| + |y| + r_x(y) = \max_{x \in C} |x| + \max_{y \in C_x} |y| + r_x(y) \\
&= \max_{x \in C} |x| + w(C_x, r_x) = \max_{x \in C} |x| + w(r(x)) \\
&= w(C, w \circ r) = w(\mathcal{C}w(C, r)).
\end{aligned}$$

For  $(C, r), (D, s) \in \mathcal{C}W$ , let us define  $(C, r) \leq (D, s)$  if there is a bijective function  $f : C \rightarrow D$  such that  $r = s \circ f$  and

$$w(C, r) \leq w(D, s).$$

**Lemma 6.**  $\mu_W : \mathcal{C}^2W \rightarrow \mathcal{C}W$  and  $\eta_W : W \rightarrow \mathcal{C}W$  are monotone with respect to  $\leq$ .

*Proof.* For  $\eta_W$ , if  $a \leq b$ , then  $w(\eta_W(a)) = a \leq b = w(\eta_W(b))$ .

For  $\mu_W$ , suppose  $(C, r), (D, s) \in \mathcal{C}^2W$  and  $(C, r) \leq (D, s)$ . According to axiom (iii),

$$\begin{aligned}
(C, r) \leq (D, s) &\Leftrightarrow \mathcal{C}w(C, r) \leq \mathcal{C}w(D, s) \\
&\Leftrightarrow w(\mathcal{C}w(C, r)) \leq w(\mathcal{C}w(D, s)).
\end{aligned}$$

Then

$$w(\mu_W(C, r)) = w(\mathcal{C}w(C, r)) \leq w(\mathcal{C}w(D, s)) = w(\mu_W(D, s)). \quad \square$$

**Theorem 7.** The algorithm of §4 for the algebra  $(\mathbb{N}, w)$  and ordering relation  $\leq$  defined by  $w$  is equivalent to determining whether an embedding of a PIFO tree in a bounded  $d$ -ary tree exists and finding the embedding if so.

## 7 Conclusion

We have presented a generalized Huffman algorithm and shown that two known algorithms, Huffman codes and embedding of PIFOs trees, can be derived as special cases. The PIFO embedding algorithm was introduced in [1] and observed to be very similar to the usual combinatorial algorithm for optimal Huffman codes, albeit based on a different algebraic structure. This suggested the common generalization presented in this paper.

Our generalized algorithm exploits the monadic structure of prefix codes, which allows a more algebraic treatment of the Huffman algorithm than the usual combinatorial approaches. The two applications fit naturally in the categorical setting by choosing specific Eilenberg-Moore algebras for each one. It is possible that other greedy algorithms might fit into this framework as well.

## References

- [1] Anshuman Mohan, Yunhe Liu, Nate Foster, Tobias Kappé, and Dexter Kozen, “Formal abstractions for packet scheduling,” Tech. Rep. <http://arxiv.org/abs/2211.11659>, Cornell University, November 2022.
- [2] Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory*, Wiley, second edition, 2006.
- [3] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown, “Programmable packet scheduling at line rate,” in *SIGCOMM*, 2016.
- [4] Albert Gran Alcoz, Alexander Dietmüller, and Laurent Vanbever, “SP-PIFO: Approximating push-in first-out behaviors using strict-priority queues,” in *NSDI*, 2020.
- [5] Andrea Asperti and Giuseppe Longo, *Categories, Types and Structures: An introduction to category theory for the working computer scientist*, Foundations of Computing. MIT Press, 1991.
- [6] Michael Barr and Charles Wells, *Toposes, Triples and Theories*, vol. 278 of *Grundlehren der mathematischen Wissenschaften*, Springer, 2013.
- [7] Michael Barr and Charles Wells, *Category Theory for Computing Science*, Prentice Hall, 1990.
- [8] Jiří Adámek, Horst Herrlich, and George E. Strecker, *Abstract and concrete categories*, Dover Publications, 2009.
- [9] Eugenio Moggi, “Notions of computation and monads,” *Inf. and Comp.*, vol. 93, no. 1, pp. 55–92, 1991.
- [10] Philip Wadler, “Comprehending monads,” *Mathematical Structures in Computer Science*, vol. 2, pp. 461–493, 1992.
- [11] Philip Wadler, “Monads for functional programming,” in *Advanced Functional Programming: 1st Int. School on Advanced Functional Programming Techniques*, Johan Jeuring and Erik Meijer, Eds., vol. 925 of *Lecture Notes in Computer Science*, pp. 24–52. Springer-Verlag, 1995.