Journal of Universal Computer Science, vol. 3, no. 11 (1997), 1207-1225
submitted: 8/8/97, accepted: 30/10/97, appeared: 28/11/97 © Springer Pub. Co.

Representing Variable-Length Codes in Fixed-Length T-Depletion Format in Encoders and Decoders[1] [2]

Ulrich Günther
(Department of Computer Science, The University of Auckland, New Zealand
Email: u.guenther@auckland.ac.nz)

Peter Hertling
(Department of Computer Science, The University of Auckland, New Zealand
Email: hertling@cs.auckland.ac.nz)

Radu Nicolescu
(Department of Computer Science, The University of Auckland, New Zealand
Email: r.nicolescu@auckland.ac.nz)

Mark Titchener
(Department of Computer Science, The University of Auckland, New Zealand
Email: mark@cs.auckland.ac.nz)

Abstract: T-Codes are a class of variable-length codes. Their self-synchronization properties are useful in compression and communication applications where error recovery rather than error correction is at issue, for example, in digital telephony. T-Codes may be used without error correction or additional synchronization mechanisms. Typically, the representation of variable-length codes is a problem in computers based on a fixed-length word architecture. This presents a problem in encoder and decoder applications. The present paper introduces a fixed-length format for storing and handling variable-length T-Code codewords, the *T-depletion codewords*, which are derived from the recursive construction of the T-Code codewords. The paper further proposes an algorithm for the conversion of T-Code codewords into T-depletion codewords that may be used as a decoder for generalized T-Codes. As well as representing all codewords of a T-Code set (the leaf nodes in the set's decoding tree), the T-depletion code format also permits the representation of "pseudo-T codewords" — strings that are not in the T-Code set. These strings are shown to correspond uniquely to all proper prefixes of T-Code codewords, thus permitting the representation of both intermediate and final decoder states in a single format. We show that this property may be used to store arbitrary finite and prefix-free variable-length codes in a compact fixed-length format.

Key Words: variable-length codes, T-Codes, encoders, decoders, representation

Category: H.3.3

---

[1] The authors' research is supported by the Department of Computer Science, the Division of Science and Technology (Tamaki Campus), and the Graduate Research Fund, all of The University of Auckland, the Centre for Discrete Mathematics and Theoretical Computer Science (CDMTCS) of the University of Auckland and the University of Waikato, and by the Deutsche Forschungsgemeinschaft (DFG).

[2] Proceedings of the *First Japan-New Zealand Workshop on Logic in Computer Science*, special issue editors D.S. Bridges, C.S. Calude, M.J. Dinneen and B. Khoussainov.

1 Introduction

Variable-length codes are often used in data compression applications (to save storage space) or data communication applications (to reduce bandwidth requirements). One of the best known variable-length code constructions was introduced by [Huffman 52]. Other variable-length coding schemes, including alphabetic encodings, are surveyed elsewhere [Gilbert and Moore 59], [Gallager 68]. The T-Codes discussed in this paper represent a subset of Huffman codes distinguished by their recursive construction.

When selecting a particular variable-length coding scheme for an application, the choice often represents a compromise between the following criteria:

- coding efficiency : encoding a message from a source with known statistics, using as few channel alphabet symbols as possible.
- complexity: implementation "cost" of the encoder and decoder.
- robustness: decodability of at least some of the encoded data in the face of errors.

This paper investigates a case where some emphasis is placed on the last two criteria — encoder/decoder cost and code robustness.

By their nature, variable-length codes are generally cumbersome to handle in computers with a fixed-length word architecture. For this reason, we require fixed-length representations for variable-length codes and conversions between variable-length code and fixed-length code formats.

For the purposes of this paper, we regard an encoder as an algorithm or device that will convert source data from a fixed-length input format into a unique variable-length output format. Similarly, we regard a decoder as an algorithm or device that performs the inverse operation, i.e., convert variable-length input data uniquely into a fixed-length output. In particular, we consider the case where a sequence of fixed-length inputs is encoded into a series of concatenated variable-length outputs, yielding a continuous symbol stream that is decoded and reconverted into a sequence of fixed-length outputs.

One way of storing variable-length codewords in a fixed-length format is to store both the codeword string and its length, where the length of the field used to store the codeword string is often determined by the length of the longest codeword in the set. An example will be given in Section 2.4, where we show that this representation is not very space-efficient compared to the representation proposed in this paper.

In decoders, such as the universal decoder for variable-length codes proposed by [Tanaka 87] or [Chung 97], an enumeration of the codewords can bypass the need to handle codewords in their literal form. However, if the codeset used is not fixed, it may need to be communicated to the decoder. In this case, a general variable-length code such as a Huffman code still requires a format similar to the one above to communicate the codewords and their "translations". Canonical Huffman codes such as treated by Hirschberg and Lelewer [Hirschberg and Lelewer 90] simplify the decoder considerably by establishing a convention that permits the derivation of the decoding tree from the code's codelength distribution. However, (canonical) Huffman codes as such are not neccessarily robust, in particular with respect to synchronization.

Synchronization is required whenever a decoder for variable-length codes decodes a wrong variable-length codeword after an error in the received symbol stream, e.g., due to noise on a communication channel. The synchronization properties of variable-length codes have therefore been of some interest [Gilbert 60],[Ferguson and Rabinowitz 84],[Wei and Scholtz 80], [Montgomery and Abrahams 86],[Takishima, Wada and Murakami 94].

The variable-length codes discussed in this paper are T-Codes [Titchener 85], which specifically derive robust self-synchronization from a recursive construction algorithm [Titchener and Hunter 85],[Titchener 95], [Günther and Titchener 95],[Günther 96].

In this paper, we propose an encoding and decoding scheme for T-Codes based on a compact fixed-length representation of the variable-length T-Code codewords, the T-depletion codewords.

The predecessor to the T-depletion codes introduced in this paper, the binary depletion codes, were initially proposed by [Titchener 85]. [Zolghadr, Honary and Darnell 89] used binary T-depletion codes to implement a real-time channel evaluation system. With the generalization of the initial T-Code construction [Titchener 95], it is now necessary to extend the notion of these depletion codes. The present paper thus develops a general formal definition for T-depletion codes that permits the representation of generalized T-Codes based on binary and non-binary alphabets.

In the following, we provide a brief introduction to T-Codes and their construction algorithm. Section 2 shows that T-Code codewords have a unique recursive structure. This structure permits us to associate a unique fixed-length codeword, the so-called T-depletion codeword with each T-Code codeword. We propose conversion algorithms between T-Code codewords and T-depletion codewords that may be used as encoders and decoders for T-Codes. The question of storage cost for T-depletion codes is shown to compare favorably with that of conventional storage schemes. Section 3 shows that the T-depletion codeword format also permits the unambiguous storage of all incomplete T-Code codewords (i.e., of prefixes of codewords which may be regarded as intermediate decoder states). The representations of complete and incomplete codewords are shown to complement each other. Finally, in Section 4, we use this property to show that any prefix-free variable-length code can be represented in a fixed-length T-depletion code format.

1.1 Generalized T-augmentation and T-Code Sets

[Titchener 85] introduced and subsequently extended [Titchener 95] a recursive construction for variable-length codes that exhibit improved self-synchronization properties. This construction is referred to as *T-augmentation*, and the resulting code sets are referred to as the T-Codes. The definition of the generalized T-Code sets is presented in [Titchener 95]. A practical introduction is provided in [Günther 96].

One might view T-augmentation as a construction which is similar to that of "escape sequences", i.e., the use of a character (an escape character) to extend the range of character commands or codes. The escape character is 'prefixed' to other characters. In this analogy, we shall allow that the escape character may be prefixed to itself to derive a further command, but is otherwise not a command in its own right.

The basic notation used in the formalization of this concept is the following: S denotes a finite alphabet and $\#S$ its cardinality. Without loss of generality, we shall presume that $S = \{0, 1, \ldots, \#S - 1\}$, i.e., that the set may be ordered and enumerated and that the symbols in S may be represented by their enumeration index k_0', where $0 \leq k_0' \leq k_0$ with k_0 defined as $k_0 = \#S - 1$. Further, let $S^\star$ be the set of all finite strings formed by concatenating the symbols in S. The empty string is denoted by λ, and we define $S^+ = S^\star \backslash \{\lambda\}$. For strings $x, y \in S^\star$, xy denotes their concatenation, and for some non-negative integer n, x^n denotes the concatenation of n copies of x, where $x^0 = \lambda$.

We are now ready to formally define T-augmentation:

Definition 1. (T-augmentation) Consider a set $X \subset S^+$, a string $p \in X$, and a positive integer k. The set Y is said to have been obtained by **T-augmentation** of X with p and k if

$$Y = \bigcup_{i=0}^{k} \{p^i s | s \in X \backslash \{p\}\} \ \cup \{p^{k+1}\} \tag{1}$$

The string p is called the **T-prefix**, and the integer k is called the **T-expansion parameter** or **T-expansion factor**.

Within the above constraints, the T-expansion parameter k and the T-prefix p may be freely chosen. In the above analogy, p corresponds to the "escape character" and k is the maximum number of "multiple escapes" permitted.

In a more compact notation, Y may also be written as $Y = [X]_{(p)}^{(k)}$.

Definition 2. (T-Code Sets) Consider a series of $n \geq 0$ successive T-augmentations of S using (for $n > 0$) the T-prefixes $p_1, p_2, \ldots, p_n$, and T-expansion parameters $k_1, k_2, \ldots, k_n$ respectively. The resulting set, denoted $S_{(p_1, p_2, \ldots, p_n)}^{(k_1, k_2, \ldots, k_n)}$, is referred to as a **T-Code set** at **T-augmentation level** n.
The T-Code sets $S_{(p_1, p_2, \ldots, p_{n'})}^{(k_1, k_2, \ldots, k_{n'})}$ where $0 \leq n' \leq n$ are referred to as the **intermediate (T-Code) sets** (of $S_{(p_1, p_2, \ldots, p_n)}^{(k_1, k_2, \ldots, k_n)}$). The vectors $(p_1, p_2, \ldots, p_n)$ and $(k_1, k_2, \ldots, k_n)$ are referred to as the **T-prefix vector** and **T-expansion vector** respectively. We also define S to be a T-Code set at T-augmentation level 0.

Example 1. (Construction of a T-Code set at T-augmentation level 3) Table 1 shows the construction and the intermediate T-Code sets for the binary ($S = \{0, 1\}$) T-Code set $S_{(0,1,01)}^{(1,1,3)}$ at the T-augmentation levels from 0 to 3.

The available choices for the T-prefixes $p_1, p_2, \ldots, p_n$ and T-expansion parameters $k_1, k_2, \ldots, k_n$ give rise to a wide range of possible sets with different cardinalities and code length distributions. Thus one may select a construction to achieve good coding efficiency, expected synchronization delay [Higgie 91][Higgie 96], or perhaps certain spectral properties for applications with a bandlimited communication channel.

For the purposes of this paper, the prefix-freeness of T-Code sets plays an important role, and we shall prove it here briefly:

Theorem 3. (Prefix-freeness of T-Code sets) *T-Code sets are prefix-free, that is, for any two codewords $x_1, x_2 \in S_{(p_1, p_2, \ldots, p_n)}^{(k_1, k_2, \ldots, k_n)}$, $x_1 y = x_2$ implies that $y = \lambda$.*

	T-augmentation level			
n	0	1	2	3
k_n	$k_0 = 1$	1	1	3
set	S	$S_{(0)}^{(1)}$	$S_{(0,1)}^{(1,1)}$	$S_{(0,1,01)}^{(1,1,3)}$
	0	$\cancel{0}$	–	–
	1	1	$\cancel{1}$	–
		00	00	00
		01	01	$\cancel{01}$
			–	–
			11	11
			100	100
			101	101
				–
				–
				0100
				$\cancel{0101}$
				–
				0111
				01100
				01101
				–
				–
				010100
				$\cancel{010101}$
				–
				010111
				0101100
				0101101
				–
				–
				01010100
				01010101
				–
				01010111
				010101100
				010101101

Table 1: The columns in the table list the codewords in the intermediate T-Code sets S (T-augmentation level 0), $S_{(0)}^{(1)}$ (T-augmentation level 1), $S_{(0,1)}^{(1,1)}$ (T-augmentation level 2) of the final set $S_{(0,1,01)}^{(1,1,3)}$ at T-augmentation level 3 listed in the last column. The table illustrates the concept of T-augmentation: a new column/T-Code set at level $i+1$ may be derived from its predecessor at level i to the left by copying the codeword list into a new column a total of $k_{i+1}+1$ times. The copies may be indexed by k'_{i+1} such that $k'_{i+1} = 0, \ldots, k_{i+1}$. Each copy is then prefixed with $p_{i+1}^{k'_{i+1}}$. Finally, all codewords of the form $p_{i+1}^{k'_{i+1}}$ are removed from the list.

Proof. by induction over n. The alphabet S is by definition prefix-free. By induction hypothesis, $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ is prefix-free. By Definition 1, the prefix-freeness of $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ implies the prefix-freeness of $S_{(p_1,p_2,\ldots,p_{n+1})}^{(k_1,k_2,\ldots,k_{n+1})}$. □

We shall make use of this property repeatedly throughout this paper.

We will now show how the T-depletion code format may be derived from the general form of T-Code codewords.

2 The Structure of T-Code Codewords and T-Depletion Codes

2.1 The Structure of T-Code Codewords

We first prove two lemmata which we will then use in the proof of the central theorem of this section, and in the proof of another theorem in Section 3.

Lemma 4. *Consider a string of the form*

$$x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}. \tag{2}$$

where $0 \leq k'_i \leq k_i$ *for* $i = 1, \ldots, n$. *Then* x *is a proper prefix of a codeword in the set* $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$, *i.e., there exists a string* $y \in S^+$ *such that* $xy \in S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$.

Proof. by induction over n. For $n = 0$, $x = \lambda$ (where λ denotes the empty string), which is clearly a proper prefix of all elements of S. Now consider

$$x = p_{n+1}^{k'_{n+1}} p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}. \tag{3}$$

By induction hypothesis, the word $p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}$ is a proper prefix of a codeword $z \in S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$. We now distinguish two cases: if $z \neq p_{n+1}$, then $p_{n+1}^{k'_{n+1}} z$ is a codeword in $S_{(p_1,p_2,\ldots,p_{n+1})}^{(k_1,k_2,\ldots,k_{n+1})}$ and hence x is a proper prefix of it. If $z = p_{n+1}$, then x is a proper prefix of $p_{n+1}^{k_{n+1}+1} \in S_{(p_1,p_2,\ldots,p_{n+1})}^{(k_1,k_2,\ldots,k_{n+1})}$. □

We can now use this result to prove the following lemma:

Lemma 5. *For any string* $x \in S^\star$ *and a given T-Code set* $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$, *there exists at most one set* $\{k'_1, \ldots, k'_n\}$ *with* $0 \leq k'_i \leq k_i$ *for* $i = 1, \ldots, n$ *such that* x *may be written in the form of Equation (2).*

Proof. by induction over n. For $n = 0$, only the empty word λ can be written in the form of Equation (2), and this representation is unique.

For the induction step we assume that a word $x \in S^\star$ can be written in two forms satisfying Equation (2):

$$x = p_{n+1}^{k'_{n+1}} p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}. \tag{4}$$

and

$$x = p_{n+1}^{k''_{n+1}} p_n^{k''_n} p_{n-1}^{k''_{n-1}} \dots p_1^{k''_1}, \tag{5}$$

such that $0 \le k'_i \le k_i$, $0 \le k''_i \le k_i$ for $i = 1, \dots, n+1$. We need to show that necessarily $k'_i = k''_i$ for all $i = 1, \dots, n+1$.

We distinguish three cases:

1. $k'_{n+1} = k''_{n+1}$: in this case, $p_n^{k'_n} p_{n-1}^{k'_{n-1}} \dots p_1^{k'_1} = p_n^{k''_n} p_{n-1}^{k''_{n-1}} \dots p_1^{k''_1}$ and the induction hypothesis applies such that $k'_i = k''_i$ for $i = 0, \dots, n$.
2. $k'_{n+1} > k''_{n+1}$: in this case, p_{n+1} is a prefix of $p_n^{k''_n} p_{n-1}^{k''_{n-1}} \dots p_1^{k''_1}$. However, by Lemma 4, this is a prefix of a codeword in $S_{(p_1,p_2,\dots,p_n)}^{(k_1,k_2,\dots,k_n)}$. Thus, p_{n+1} would have to be a prefix of a codeword in $S_{(p_1,p_2,\dots,p_n)}^{(k_1,k_2,\dots,k_n)}$, which violates the prefix-freeness of that set.
3. $k'_{n+1} < k''_{n+1}$: this case also violates the prefix-freeness of $S_{(p_1,p_2,\dots,p_n)}^{(k_1,k_2,\dots,k_n)}$, by the same argument as before.

This proves Lemma 5. □

We now prove the central theorem of this section: the existence of a unique form for T-Code codewords in $S_{(p_1,p_2,\dots,p_n)}^{(k_1,k_2,\dots,k_n)}$:

Theorem 6. (Decomposition of T-Code Codewords) *For all codewords $x \in S_{(p_1,p_2,\dots,p_n)}^{(k_1,k_2,\dots,k_n)}$, a decomposition*

$$x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \dots p_1^{k'_1} k'_0, \tag{6}$$

exists such that $0 \le k'_i \le k_i$ for $i = 0, 1, \dots, n$ and $k_0 = \#S - 1$. The decomposition of x is unique, i.e., there exists exactly one set of k'_i for which Equation (6) is satisfied.

Proof. **Existence**: by induction over n. For $n = 0$, the T-Code set is the alphabet S itself. As we may represent each alphabet symbol by an integer $0 \le k'_0 \le k_0$, codewords x_0 at T-augmentation level 0 are of the general form $x = k'_0$, which satisfies the theorem.

By the induction hypothesis, the codewords of $S_{(p_1,p_2,\dots,p_n)}^{(k_1,k_2,\dots,k_n)}$ satisfy the general form

$$x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \dots p_1^{k'_1} k'_0, \tag{7}$$

for some k'_i, $i = 0, 1, \dots, n$, such that $0 \le k'_i \le k_i$. The T-augmentation of $S_{(p_1,p_2,\dots,p_n)}^{(k_1,k_2,\dots,k_n)}$ with p_{n+1} and k_{n+1} prefixes these codewords with $k'_{n+1} \le k_{n+1}$ copies of p_{n+1}. The codewords in $S_{(p_1,p_2,\dots,p_n,p_{n+1})}^{(k_1,k_2,\dots,k_n,k_{n+1})}$ thus satisfy the general form

$$x = p_{n+1}^{k'_{n+1}} p_n^{k'_n} p_{n-1}^{k'_{n-1}} \dots p_1^{k'_1} k'_0. \tag{8}$$

Thus, we have shown the existence of the decomposition.

Uniqueness: write a codeword $x \in S_{(p_1,p_2,\dots,p_n)}^{(k_1,k_2,\dots,k_n)}$ in the form of Equation 6:

$$x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \dots p_1^{k'_1} k'_0. \tag{9}$$

The uniqueness of k'_0 is clear as it is simply the last symbol in x. The uniqueness of $k'_1, \ldots, k'_n$ follows from Lemma 5 by applying it to the string $p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}$. This concludes the proof of Theorem 6. $\square$

In fact, the representation in form of Equation (6) is unique not only for codewords in $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$, but also for all codewords in its intermediate T-Code sets:

Theorem 7. (Decomposition for Intermediate T-Code Sets) *For any $m \leq n$, the decomposition of $x \in S_{(p_1,p_2,\ldots,p_m)}^{(k_1,k_2,\ldots,k_m)}$ as*

$$x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} k'_0 \tag{10}$$

with $0 \leq k'_i \leq k'_i$ exists and is unique. It satisfies $k'_i = 0$ for $i > m$.

Proof. The existence of a decomposition with $k'_i = 0$ for $i > m$ follows from Theorem 6 for $k'_i = 0$ for $i > m$. Again, the uniqueness of k'_0 is clear. The uniqueness of $k'_1, \ldots, k'_n$ follows from Lemma 5. $\square$

Definition 8. (T-expansion indices and literal symbol) Let $x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} k'_0$ with $x \in S_{(p_1,p_2,\ldots,p_m)}^{(k_1,k_2,\ldots,k_m)}$ for $m \leq n$ and $k'_i \leq k_i$ for $i = 0, \ldots, n$. For $i \geq 1$, we call k'_i the i'th **T-expansion index** of x. k'_0 is called the **literal symbol**.

Example 2. (Decomposition of T-Code codewords) Table 2 shows the decomposition of the codewords from the binary ($S = \{0,1\}$) T-Code set $S_{(0,1,01)}^{(1,1,3)}$.

[Nicolescu 95] showed that the decomposition of one of the longest codewords in a T-Code set may be used to represent the set itself. The longest codewords contain all T-prefixes, and their T-expansion indices equal the T-expansion factors of the corresponding set. Hence, it is possible to communicate the whole T-Code set to a decoder by simply sending one of the longest codewords[3].

2.2 T-Depletion Codes

Presume that for a given T-Code set the T-prefixes and T-expansion parameters are known. Thus, we may specify any codeword in such a set set simply by stating the corresponding T-expansion indices $k'_1, k'_2, \ldots, k'_n$ and literal symbol k'_0. For example, if the T-Code set $S_{(0,1,01)}^{(1,1,3)}$ from the previous examples was given, and we specified that $k'_0 = 1$, $k'_1 = 0$, $k'_2 = 1$, and $k'_3 = 2$, it would be unambiguous that this combination would refer to $(01)^2 1^1 0^0 1 = 010111$. T-depletion codewords, which may be defined in terms of multibase numbers, implement this format:

[3] The T-prefixes and T-expansion parameters cannot necessarily be uniquely determined this way. Consider, e.g., $S_{(0)}^{(3)} = S_{(0,00)}^{(1,1)}$ for $S = \{0, 1\}$.

T-Code	structure
00	$(01)^0 1^0 0^1 0$
11	$(01)^0 1^1 0^0 1$
100	$(01)^0 1^1 0^1 0$
101	$(01)^0 1^1 0^1 1$
0100	$(01)^1 1^0 0^1 0$
0111	$(01)^1 1^1 0^0 1$
01100	$(01)^1 1^1 0^1 0$
01101	$(01)^1 1^1 0^1 1$
010100	$(01)^2 1^0 0^1 0$
010111	$(01)^2 1^1 0^0 1$
0101100	$(01)^2 1^1 0^1 0$
0101101	$(01)^2 1^1 0^1 1$
01010100	$(01)^3 1^0 0^1 0$
01010101	$(01)^3 1^0 0^1 1$
01010111	$(01)^3 1^1 0^0 1$
010101100	$(01)^3 1^1 0^1 0$
010101101	$(01)^3 1^1 0^1 1$

Table 2: Codeword decomposition for the T-Code set $S_{(0,1,01)}^{(1,1,3)}$.

Definition 9. (Multibase Numbers) A vector $(k'_n, k'_{n-1}, \ldots, k'_1, k'_0)$ is called a **multibase number** with base $(k_n+1, k_{n-1}+1, \ldots, k_1+1, k_0+1)$ if for all i, $0 \le i \le n$

$$0 \le k'_i \le k_i. \tag{11}$$

Definition 10. (T-Depletion Codewords) Given a T-Code set $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$, a multibase number $(k'_n, k'_{n-1}, \ldots, k'_1, k'_0)$ is called the **T-depletion codeword** $d_n(x)$ corresponding to $x \in S_{(p_1,p_2,\ldots,p_m)}^{(k_1,k_2,\ldots,k_m)}$ if $m \le n$, $0 \le k'_i \le k_i$ for $i = 0, \ldots, n$, and

$$x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} k'_0. \tag{12}$$

The notion of multibase numbers will probably be familiar to the reader: for example, a 24-hour clock may be represented by a multibase number with base $(24, 60, 60)$, i.e., one index running from 0 to 23 for the hours, and two indices running from 0 to 59 each for the minutes and seconds.

Note that, by Theorem 7, a unique **T-depletion codeword** exists for each codeword from the intermediate T-Code sets $S_{(p_1,p_2,\ldots,p_m)}^{(k_1,k_2,\ldots,k_m)}$.

T-depletion codewords are illustrated in the following example:

Example 3. The codewords from the binary ($S = \{0, 1\}$) T-Code set $S_{(0,1,01)}^{(1,1,3)}$ and their associated T-depletion codewords are listed in Table 3.

Since the decomposition of T-Code codewords reflects the recursive construction of the T-Codes, the T-depletion codewords reflect it, too: consider, for example, a T-depletion codeword $d_n(x) = (k'_n, k'_{n-1}, \ldots, k'_1, k'_0)$ of a T-Code codeword

T-Code	T-depletion codeword
	(k'_3, k'_2, k'_1, k'_0)
00	$(0, 0, 1, 0)$
11	$(0, 1, 0, 1)$
100	$(0, 1, 1, 0)$
101	$(0, 1, 1, 1)$
0100	$(1, 0, 1, 0)$
0111	$(1, 1, 0, 1)$
01100	$(1, 1, 1, 0)$
01101	$(1, 1, 1, 1)$
010100	$(2, 0, 1, 0)$
010111	$(2, 1, 0, 1)$
0101100	$(2, 1, 1, 0)$
0101101	$(2, 1, 1, 1)$
01010100	$(3, 0, 1, 0)$
01010101	$(3, 0, 1, 1)$
01010111	$(3, 1, 0, 1)$
010101100	$(3, 1, 1, 0)$
010101101	$(3, 1, 1, 1)$

Table 3: Codewords and T-depletion code elements for the T-Code set $S_{(0,1,01)}^{(1,1,3)}$.

$x \in S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$, i.e., $x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} k'_0$. We may thus split x recursively into k'_n copies of the n'th level T-prefix p_n, and the codeword $p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} k'_0$, all of which are codewords in $S_{(p_1,p_2,\ldots,p_{n-1})}^{(k_1,k_2,\ldots,k_{n-1})}$, etc.

Similarly, the T-depletion codeword $d_n(x)$ may be interpreted as accounting for k'_n copies of $d_{n-1}(p_n)$, and one copy of $d_{n-1}(p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} k'_0)$.

This may be utilized in the construction of encoders and decoders, which may be regarded as converters between T-Code and T-depletion codewords, i.e., between the variable-length format and the fixed-length format.

2.3 Conversion between Variable-Length T-Code Codewords and T-Depletion Codewords

In Table 3, we listed the T-Code codewords from $S_{(0,1,01)}^{(1,1,3)}$ and their corresponding T-depletion codewords, but we did not explain how we obtained one format from the other. We will now discuss these conversion issues for both directions:

2.3.1 Encoders: Converting T-Depletion into T-Code Codewords

Theorem 6 provides us with an easy way of converting T-depletion codewords into their variable-length T-Code codewords counterparts in $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$. We only need to concatenate the appropriate number of T-prefixes and the literal symbol. Since all T-prefixes and the literal symbol may be found in at least one of the intermediate sets $S^{(k_1,k_2,\ldots,k_m)}_{(p_1,p_2,\ldots,p_m)}$, $m < n$, each of these strings has a T-depletion code associated with it. Hence the concatenation of the T-prefixes and the literal symbol may be done recursively.

If a T-Code set is to be communicated to, or to be stored by, a decoder in a fixed-length format, this may thus be achieved by representing it by the T-depletion codewords for its T-prefixes and their respective T-expansion parameters. This information suffices to permit the recursive encoding or decoding of every codeword in the set.

2.3.2 Decoders: Converting T-Codes into T-Depletion Codewords

The conversion in the other direction, i.e., obtaining a T-depletion codeword from a variable-length T-Code codeword, may be used in decoders. The pseudo-code in Figure 1 implements such a recursive decoder. A T-Code codeword at T-augmentation level n is read left-to-right as a concatenation of codewords from the intermediate set at level $n-1$. This concatenation sequence consists of up to k_n T-prefixes and another codeword from the intermediate set. To obtain the number of T-prefixes k'_n and the significant T-depletion codeword elements for the other codeword from the intermediate set at level $n-1$, the decoder route calls itself recursively as a decoder over the intermediate set. Only at level 0, actual symbols from the decoder's input are read.

Example 4. The T-depletion codeword format for codewords from $S^{(1,1,3)}_{(0,1,01)}$ is (k'_3, k'_2, k'_1, k'_0). We follow the entries in this format as we decode the codeword 01010101 from $S^{(1,1,3)}_{(0,1,01)}$. The following "snapshots" of the global T-depletion codeword register may be taken at the point indicated in the pseudo-code listing in Figure 1:

- initial state: $(-,-,-,-)$, not initialized. As the procedure calls itself recursively, the T-expansion indices are initialized top-down, i.e., $(0,-,-,-)$, then $(0,0,-,-)$, $(0,0,0,-)$, and $(0,0,0,0)$.
- decode $x[1]=0$: "snapshot" $(0,0,0,0)$. Upon return to the calling procedure, it is determined that the (0) is a copy of p_1, i.e., $(0)=d_0(p_1)=(0)$. Thus k'_1 is incremented to $k'_1=1$. The next run through the loop resets k'_0 to 0.
- decode $x[2]=1$: "snapshot" $(0,0,1,1)$. Upon return to the calling procedure, it is determined that $(1)\neq d_0(p_1)=(0)$. Thus, the procedure itself returns to its own calling level, where $(1,1)$ is determined not to be a copy of p_2, i.e., $(1,1)\neq d_1(p_2)=(0,1)$, and program execution returns to the next higher level. There, it is found that $(0,1,1)=d_2(p_3)$, and k'_3 is incremented to $k'_3=1$. In the next run of the loop, k'_2, k'_1, and k'_0 are successively reset to 0: $(1,0,0,0)$
- decode $x[3]=0$: "snapshot" $(1,0,0,0)$, another copy of p_1. Increment k'_1 to $k'_1=1$ and reset k'_0: $(1,0,1,0)$
- decode $x[4]=1$: "snapshot" $(1,0,1,1)$, $(1,1)\neq d_0(p_1)$ and $(1,1)\neq d_0(p_2)$, but $(0,1,1)=d_2(p_3)$, i.e., we have found another copy of p_3. Increment k'_3 to $k'_3=2$ and reset k'_0, k'_1, k'_2: $(2,0,0,0)$.
- decode $x[5]=0$: "snapshot" $(2,0,0,0)$, a copy of p_1. Increment k'_1 and reset k'_0

```
program conversion;
var
    global x: string;
    global i: integer;
    global (k'_n, k'_{n-1}, ..., k'_1, k'_0): TDepletionCodeWord;

procedure tconvert(m: integer);
begin
    k'_m := 0;                          {clear T-expansion index}
    if ( m > 0 )
    then begin {loop}                   {decode at lower level}
        {check if next lower-level codeword is p_m}
        tconvert(m - 1);                {decode next lower-level codeword}
        {check if it matches the T-depletion codeword for p_m by comparing}
        {the new T-expansion indices with the T-depletion codeword for p_m:}
        if ( (k'_m, k'_{m-1}, ..., k'_1, k'_0) ≠ d_{m-1}(p_m) )
        then break;                     {it is not p_m}
        {if we do not break here, the T-expansion indices found match p_m.}
        {However, if it is the (k_m + 1)'th copy of p_m, it does not count as}
        {a T-prefix and we have found the end of a codeword at level m:}
        if ( k'_m < k_m )
        then                            {it is a genuine copy of the T-prefix p_m}
            k'_m := k'_m + 1;           {thus increment T-expansion index k'_m}
        else                            {it is the (k_m + 1)'th copy:}
            break;                      {end of codeword at level m found}
    end {loop}
    else begin                          {symbol level reached - read next symbol}
        {The snapshots in Example 4 are taken at this point}
        i := i + 1;                     {advance string pointer to next symbol}
        k'_0 := x[i];                   {read next symbol}
    end;
end;

begin
    i := 0;                             {initialize string pointer}
    x :=ATCodeCodeWord;                 {this is the codeword we want to convert}
    tconvert(n);                        {run conversion}
    output((k'_n, k'_{n-1}, ..., k'_1, k'_0)); {output result}
end.
```

Figure 1: A T-Code decoder: the pseudo-code routine above converts the T-Code codeword x from $S_{(p_1,p_2,\dots,p_n)}^{(k_1,k_2,\dots,k_n)}$ into a T-depletion codeword $d_n(x) = (k'_n, k'_{n-1}, \dots, k'_1, k'_0)$ by decoding it with a recursive decoding routine. Note that the T-expansion indices k'_i are determined in decreasing order of i, which permits to use a single storage register (global variable) in this recursive routine.

- decode $x[6] = 1$: "snapshot" $(2, 0, 1, 1)$. As before, increment k'_3 and reset k'_0, k'_1, k'_2. Now, $k'_3 = k_3$, i.e., no more copies of p_3 can follow.
- decode $x[7] = 0$: "snapshot" $(3, 0, 0, 0)$, a copy of p_1. Increment k'_1 and reset k'_0 as before: $(3, 0, 1, 0)$.
- decode $x[8] = 1$: "snapshot" $(3, 0, 1, 1)$, as $k'_3 = k_3$, the 1 must be the literal, i.e., $k'_0 = 1$. The program returns to its top level and outputs $(3, 0, 1, 1)$. We combine $p_3^3 p_2^0 p_1^1 k'_0 = (01)^3 1^0 0^1 1 = 01010101$ to verify our result.

In a decoder, we may use the fixed-length representation of the final T-depletion codeword output as an address into a look-up table containing the (fixed-length) decoding of the corresponding T-Code codeword.

2.4 Storage Resource Requirements for T-Depletion Codewords

Conventional encoders often require the storage of both the codeword string and length. In these cases, the length of the field used to store the codeword string is often based on the length of the longest codeword in the code set. This format permits the representation of both final and intermediate encoder states: an incompletely encoded codeword may be represented by the full codeword string accompanied by a length value that is shorter than the string. Let us consider the storage cost associated with a codeword:

Example 5. Consider the codeword 01010111 from a variable-length binary code whose longest codeword is, e.g., nine bits long. The codeword may be represented in a fixed-length binary register by, e.g., a four bit length field and the codeword string itself, padded with 0's to the right (the dot indicates the border between length field and the padded string):

1000.010101110

A partially encoded (or transmitted) codeword, e.g., its first five bits, can be represented as follows:

0101.010101110

As multibase numbers, T-depletion codewords for a given T-Code set may also be stored in a fixed-length format. The storage resource requirement of this format is given by the total number of m-ary register cells required to store all entries of the multibase number. The storage of a single entry with base $(k_i + 1)$ requires $\lceil \log_m(k_i + 1) \rceil$ m-ary register cells. To store an arbitrary T-depletion codeword from $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ in an m-ary register, we hence require

$$L = \sum_{i=0}^{n} \lceil \log_m(k_i + 1) \rceil \qquad (13)$$

m-ary register cells. In the equation above, $\lceil y \rceil$ denotes the smallest integer that is larger than or equal to y, and $(k_i + 1)$ is the i'th base of our multibase number.

Example 6. Storage Requirements for T-Depletion Codewords Consider the binary set $S_{(0,1,01)}^{(1,1,3)}$ from the previous examples, and presume that we wish to store a T-depletion codeword from this set in a binary $(m = 2)$ register. Then $L = 5$, i.e., we require a 5-bit register.

If we used the format from Example 5 to store one of the codewords in $S^{(1,1,3)}_{(0,1,01)}$, we would require nine bits to store the string, and another four to store the length information, i.e., a total of thirteen bits.

Note that if $\log_m(k_i+1) \notin \mathbb{N}$ for some i, there is some inherent inefficiency in the m-ary representation of the multibase number $(k'_n, k'_{n-1}, \ldots, k'_1, k'_0)$. However, this inefficiency is always less than $n+1$ digits in the m-ary representation.

The **binary T-depletion codewords** used in [Titchener 85], [Higgie 91], [Zolghadr, Honary and Darnell 89] implicitly assume that $S = \{0,1\}$, $k'_n = k'_{n-1} = \ldots = k'_1 = 1$, and $m = 2$. In this case, the presence or absence of the T-prefix p_i in a T-Code codeword is indicated by a single bit in the T-depletion codeword. A T-Code codeword from such a **simple T-Code set** at T-augmentation level n may thus be represented by a binary register with $n+1$ bits.

However, the multibase number format associated with the T-depletion codewords also permits the representation of multibase numbers that do not correspond to T-Code codewords. As we shall see in the next section, these numbers may be interpreted as incomplete codewords, i.e., intermediate decoder states. This is discussed in the following section.

3 Pseudo-T Codewords

The astute reader will have noticed that Theorem 6 demands that T-Code codewords are of a certain form. However, it does not claim that all strings of this form are necessarily T-Code codewords. Given a T-Code set $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$, consider a string x for which k'_i exist such that $0 \leq k'_i \leq k_i$ and such that x may be written in the form given by Equation (6).

For example, the string $x = 01010110$ satisfies Equation (6) for $S^{(1,1,3)}_{(0,1,01)}$ if it is written as $x = (01)^3 1^1 0^0 0$. Such strings may be represented by the T-depletion code format. However, as is evident from Table 1, $x \notin S^{(1,1,3)}_{(0,1,01)}$. Such strings are called "pseudo-T codewords":

Definition 11. (Pseudo-T Codewords) Strings $x \notin S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ that may be written in the form of Equation (6) for some k'_i with $0 \leq k'_i \leq k_i$ for $i = 0, \ldots, n$ are called **pseudo-T codewords** for $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$. We also define the empty string λ to be a pseudo-T codeword. The set of all pseudo-T codewords for $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ is denoted $\Phi\left(S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}\right)$.

This definition provides for the existence of pseudo-T codewords, but it tells us little about their properties. The following theorem provides us with an alternative to this definition and an insight into the structure of pseudo-T codewords.

Theorem 12. (Decomposition of Pseudo-T Codewords) *The set of all pseudo-T codewords for* $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ *may be written as*

$$\Phi\left(S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}\right) = \{x | x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} \lambda, 0 \leq k'_i \leq k_i, i = 1, \ldots, n\}. \quad (14)$$

Proof. by induction over n. We note that for $n = 0$, where $\Phi(S) = \{\lambda\}$, the assertion is true.

We will now prove that

$$\Phi\left(S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}\right) \subseteq \{x|x = p_{n+1}^{k'_{n+1}} p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}\lambda, 0 \le k'_i \le k_i, i = 1,\ldots,n{+}1\}$$

and

$$\Phi\left(S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}\right) \supseteq \{x|x = p_{n+1}^{k'_{n+1}} p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}\lambda, 0 \le k'_i \le k_i, i = 1,\ldots,n{+}1\}.$$

"$\subseteq$": let $x \in \Phi\left(S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}\right)$. We wish to show that there are k'_i with $0 \le k'_i \le k_i$ such that $x = p_{n+1}^{k'_{n+1}} p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}$.

If $x = \lambda$, then this is true if $k'_i = 0$ for all i. If $x \ne \lambda$, then by Definition 11, x satisfies Equation (6) such that

$$x = p_{n+1}^{k''_{n+1}} p_n^{k''_n} p_{n-1}^{k''_{n-1}} \ldots p_1^{k''_1} k''_0. \tag{15}$$

We now distinguish two cases:

1. $p_n^{k''_n} p_{n-1}^{k''_{n-1}} \ldots p_1^{k''_1} k''_0 \in S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$: since $x \notin S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}$, Equation (1) implies that this is only possible if $k''_{n+1} < k_{n+1}$ and $p_n^{k''_n} p_{n-1}^{k''_{n-1}} \ldots p_1^{k''_1} k''_0 = p_{n+1}$. In this case, we may choose $k'_{n+1} = k''_{n+1} + 1$ and $k'_i = 0$ for $i \le n$ to satisfy the assertion.
2. $p_n^{k''_n} p_{n-1}^{k''_{n-1}} \ldots p_1^{k''_1} k''_0 \in \Phi\left(S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}\right)$: by induction hypothesis, there are k'_i such that
$$p_n^{k''_n} p_{n-1}^{k''_{n-1}} \ldots p_1^{k''_1} k''_0 = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} \tag{16}$$
and hence
$$x = p_{n+1}^{k''_{n+1}} p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} \tag{17}$$
which also satisfies the assertion.

"$\supseteq$": we have to show that any word $x = p_{n+1}^{k'_{n+1}} p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}$ with $0 \le k'_i \le k_i$ for $i \le n+1$ is not in $S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}$ but nevertheless is either the empty string λ or satisfies Equation (6) for some $0 \le k''_i \le k_i$ for $i \le n+1$. Lemma 4 and the prefix-freeness of $S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}$ guarantee that $x \notin S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}$.

If $k'_i = 0$ for all i, then $x = \lambda$ and the assertion is trivially satisfied. If $k'_i > 0$ for some i, then define m such that $k'_m > 0$ and $k'_i = 0$ for all $i < m$. Since $p_m \in S^{(k_1,k_2,\ldots,k_{m-1})}_{(p_1,p_2,\ldots,p_{m-1})}$, we may use Theorem 6 to write

$$p_m = p_{m-1}^{k''_{m-1}} \ldots p_1^{k''_1} k''_0 \tag{18}$$

for some k''_i with $0 \le k''_i \le k_i$ for $i = 0,\ldots,m-1$. Now set $k''_m = k'_m - 1$ and $k''_i = k'_i$ for $i > m$ to satisfy Equation (6).

This concludes the proof of Theorem 12. □

Together, Theorems 6 and 12 yield an alternative expression for the T-Code set $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ in a non-recursive form:

$$S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)} = \bigcup_{\substack{k'_1 \leq k_1,\ldots \\ \ldots,k'_n \leq k_n, \\ k'_0 \in S}} \{p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} k'_0\} \setminus \bigcup_{\substack{k'_1 \leq k_1,\ldots \\ \ldots,k'_n \leq k_n}} \{p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1}\}. \tag{19}$$

What is the significance of the pseudo-T codes? Comparing Equation (14) with Equation (6), we find that for all $k'_0 \in S$ and $x \in \Phi\left(S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}\right)$, strings of the form xk'_0 satisfy the general form of T-Code codewords in Equation (6). This motivates the following theorem:

Theorem 13. (Pseudo-T Codewords are Proper Prefixes of T-Codes) *Every pseudo-T codeword for $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ is a proper prefix of a T-Code codeword in $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$. Conversely, every proper prefix of a T-Code codeword in $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ is a pseudo-T codeword for $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$. I.e.,*

$$\Phi\left(S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}\right) = \left\{x \in S^\star | \exists y \in S^+ \ s.t. \ xy \in S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}\right\} \tag{20}$$

Proof. the inclusion

$$\Phi\left(S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}\right) \subseteq \left\{x \in S^\star | \exists y \in S^+ \text{ s.t. } xy \in S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}\right\}$$

follows from Theorem 12 and Lemma 4. The inclusion

$$\Phi\left(S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}\right) \supseteq \left\{x \in S^\star | \exists y \in S^+ \text{ s.t. } xy \in S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}\right\}$$

may be proven by induction as follows:

Let x be T-Code codeword in $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$, and let $x[1:i]$ for $i = 0, \ldots, |x|-1$ be its proper prefix comprising the first i symbols in x, where $x[1:0] = \lambda$.

By Theorem 6, x has the form of Equation (6) and thus $x[1:|x|-1]$ is a pseudo-T codeword by Theorem 12.

By the induction hypothesis, let $x[1:i]$ with $|x[1:i]| > 0$ be a pseudo-T codeword. Then, by Definition 11, it may be written in the form of Equation (6). Thus, by Theorem 12, $x[1:i-1]$ is a pseudo-T codeword. □

Every pseudo-T-Code codeword may be written in the form of Equation (2) and, with the exception of the empty word λ, every pseudo-T-Code codeword may be written in the form of Equation (6). It follows from Lemma 5 that both of these forms are unique.

A T-Code set, like e.g., a Huffman code, may be represented by a rooted graph (decoding tree). In such a decoding tree, the T-Code codewords thus occupy the leaf nodes (terminal nodes) of the tree, whereas the pseudo-T codewords represent all the internal (branch) nodes of the decoding tree. As a decoder traverses the tree, it will encounter these branching nodes. Pseudo-T codewords may thus be interpreted as intermediate decoder states that occur in an incomplete decoding. In fact, the reader may wish to verify from the "snapshots" in Example 4 that this is in fact the case for the decoder algorithm presented above. Also, all T-prefixes of $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ are among its pseudo-T codewords. As we shall see in the next section, this enables us to represent any finite and prefix-free variable-length code by a T-depletion codeword.

4 Pseudo-T Codewords and Variable-Length Codes

In the previous sections, we have shown that any prefix of a T-Code codeword may be represented in a T-depletion codeword format. From this, we may conclude that any prefix-free variable-length code C can be represented in such a way, provided there exists a T-Code set $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ such that all codewords in C are prefixes of codewords in $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$. But does such a T-Code set always exist?

Definition 14. (Covering T-Code Sets) Let $C, C' \subset S^+$ be finite and prefix-free variable-length codes. C' is said to be a **covering set** of C iff

$$\forall x \in C \exists y \in S^\star, z \in C' \text{ s.t. } xy = z. \tag{21}$$

In this case, we also say that C' *covers* C.

The concept of covering code sets may be visualized if one considers the decoding trees of the two sets involved.

Theorem 15. (Existence of Covering T-Code Sets) *For any finite and prefix-free variable-length code $C \subset S^+$, there exists a T-Code set $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ that covers C.*

Proof. by showing how $S_{(p_1,p_2,\ldots,p_n)}^{(k_1,k_2,\ldots,k_n)}$ may be found for a given C.

1. start with S at T-augmentation level $m = 0$, and define some lexicographical order on the elements of C.
2. if $S_{(p_1,p_2,\ldots,p_m)}^{(k_1,k_2,\ldots,k_m)}$ covers C, finish.
3. from the lexicographical order on C, select the first element $x \in C$ that is not a prefix of any codeword in $S_{(p_1,p_2,\ldots,p_m)}^{(k_1,k_2,\ldots,k_m)}$. Since $S_{(p_1,p_2,\ldots,p_m)}^{(k_1,k_2,\ldots,k_m)}$ is complete and prefix-free, there exists a unique codeword $x' \in S_{(p_1,p_2,\ldots,p_m)}^{(k_1,k_2,\ldots,k_m)}$ such that x' is a proper prefix of x. Set $p_{m+1} = x'$, $k_{m+1} = 1$, and T-augment to $S_{(p_1,p_2,\ldots,p_{m+1})}^{(k_1,k_2,\ldots,k_{m+1})}$. Increment m and continue with step 2.

This algorithm is guaranteed to terminate, because

- in the third step, $|x|$ (the length of x) is finite. However, since T-Code sets are complete, the length of x' in step 3 increases with each T-augmentation (as long as x is not a prefix of x'). Thus, $|x'|$ will eventually reach or exceed $|x|$, such that x becomes a prefix of x', i.e., the number of T-augmentations for a given x is finite.

– there are also only finitely many strings in C, such that the total number of T-augmentations must be finite.

This proves the existence of covering T-Code sets. □

It is obvious that there is no unique covering T-Code set for a given C. For example, any T-augmentation of a covering T-Code set yields another covering T-Code set. Also, restricting T-expansion parameters to a value of 1, or using a particular lexicographical order on C in the above algorithm are in principle arbitrary constraints which, if changed, may lead to vastly different solutions.

Under "storage cost" considerations, the covering T-Code set of our choice would obviously be a T-Code set for which the storage cost of the T-depletion code format is minimized. The algorithm proposed in the proof above, however, does not work optimally in this sense. This is illustrated by the following example:

Example 7. Let $S = \{0, 1\}$ be the binary alphabet. Consider the code

$$C = \{0000, 0001, 00100, 00101, 0011, 01, 100, 101, 11\}.$$

C is actually a T-Code set, $S^{(1,1,1)}_{(0,1,00)}$, and from Section 2.4 we know that its T-depletion codewords require 4 bits of storage. However, using the above algorithm (presuming a lexicographical ordering of the codewords in C as listed above), we obtain the covering T-Code set $S^{(1,1,1,1)}_{(0,00,001,1)}$, which requires 5 bits of storage.

An "optimal" algorithm in this sense remains an open problem.

5 Conclusion

In this paper, we have proposed encoder and decoder algorithms for T-Codes, based on T-depletion codewords. The algorithms permit the use of the same efficient fixed-length representation of T-Code codewords in both encoder and decoder. The size of the T-depletion codeword for a given set is a function of the T-augmentation level (the recursive depth of the T-Code set's construction), and the T-expansion parameter values. The T-depletion codewords for the T-prefixes of a T-Code set together with the corresponding expansion parameters may also be used to communicate or store a T-Code set efficiently in a fixed-length environment[4].

As shown in Section 3, the T-depletion codeword format caters not only for T-Code codewords but also for pseudo-T codewords. These may be regarded as representations of all branching nodes in a T-Code decoding tree, i.e, as incomplete codewords or intermediate decoder states. The T-depletion code format thus represents the complete state space of a decoder. In Section 4 we have shown that this concept may be extended to cover prefix-free variable-length codes in general. This presents a novel way of storing such codes in a fixed-length format.

Acknowledgement

The authors would like to thank Cris Calude for his helpful comments during the preparation of this paper.

References

1. [Chung 97] K.-L. Chung. Efficient Huffman decoding. *Information Processing Letters*, 61:97–99, 1997.
2. [Ferguson and Rabinowitz 84] T. J. Ferguson and J. H. Rabinowitz. Self synchronizing Huffman codes. *IEEE Trans. Inform. Theory*, 30(4):687–693, July 1984.
3. [Gallager 68] R. G. Gallager. *Information Theory and Reliable Communications.* John Wiley and Sons, Inc, November 1968.
4. [Gilbert 60] E. N. Gilbert. Synchronization of binary messages. *IRE Trans. Inform. Theory*, 10:933–967, 1960.
5. [Gilbert and Moore 59] E. N. Gilbert and E. F. Moore. Variable length binary encodings. *Bell Syst. Tech. J.*, 38:933–967, July 1959.
6. [Günther 96] U. Günther. Data compression and serial communication with generalized T-Codes. *Journal of Universal Computer Science*, 2(11):769–795, November 1996.
7. [Günther and Titchener 95] U. Günther and M. R. Titchener. Calculating the expexted synchronization delay for T-Code Sets. *IEE Proceedings - Communications*, 144:121, June 1997.
8. [Higgie 91] G. R. Higgie. *Analysis of the Families of Variable-Length Self-Synchronizing Codes Called T-Codes.* PhD thesis, The University of Auckland, 1991.
9. [Higgie 96] G. R. Higgie. Database of best T-Codes. *IEE Proceedings, Computers and Digital Techniques*, 143:213–218, July 1996.
10. [Hirschberg and Lelewer 90] D. S. Hirschberg and D. A. Lelewer. Efficient decoding of prefix codes. *Communications of the ACM*, 33(4):449–459, April 1990.
11. [Huffman 52] D. Huffman. A method for the construction of minimum redundancy codes. *Proc. Inst. Radio Eng.,*, 40:1098–1101, September 1952.
12. [Montgomery and Abrahams 86] B. L. Montgomery and J. Abrahams. Synchronization of binary source codes. *IEEE Trans. Inform. Theory*, 32(6):849–854, November 1986.
13. [Nicolescu 95] R. Nicolescu. Uniqueness theorems for T-Codes. Technical Report 9, The University of Auckland, September 1995.
14. [Takishima, Wada and Murakami 94] Y. Takishima, M. Wada, and H. Murakami. Error states and synchronization recovery for variable length codes. *IEEE Transactions on Communications*, 42(2-4):783–792, February 1994.
15. [Tanaka 87] H. Tanaka. Data structure of Huffman codes and its application to efficient encoding and decoding. *IEEE Trans. Inform. Theory*, 33(1):154–156, January 1987.
16. [Titchener 85] M. R. Titchener. Construction and properties of the augmented and binary-depletion codes. *IEE Proceedings Pt.E, Computers and Digital Techniques*, 132(3):163–169, May 1985.
17. [Titchener 95] M. R. Titchener. Generalized T-Codes: an extended construction algorithm for self-synchronizing variable-length codes. *IEE Proceedings Pt.E, Computers and Digital Techniques*, 143(3):122–128, June 1996.
18. [Titchener and Hunter 85] M. R. Titchener and J. J. Hunter. Synchronization process for the variable-length T-Codes. *IEE Proceedings Pt.E, Computers and Digital Techniques*, 133(1):54–64, 1985.
19. [Wei and Scholtz 80] V. K. W. Wei and R. A. Scholtz. On the characterization of statistically synchronizable codes. *IEEE Trans. Inform. Theory.*, 26(6):733–735, November 1980.
20. [Zolghadr, Honary and Darnell 89] F. Zolghadr, B. Honary and M. Darnell. Statistical real-time channel evaluation (SRTCE) technique using variable-length T-Codes. *IEE Proceedings*, 136(4):259–266, August 1989.