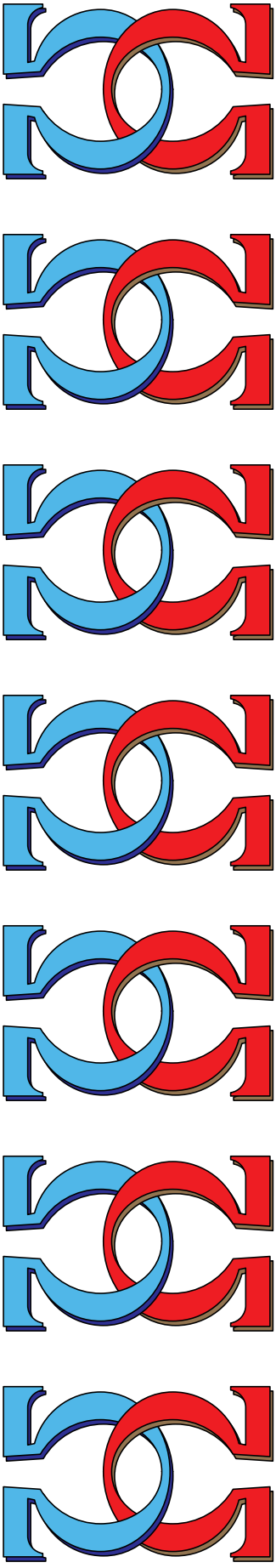


CDMTCS
Research
Report
Series

On T-codes and necklaces

T. Aaron Gulliver[1] and Ulrich Speidel[2]
[1]University of Victoria,
[2]University of Auckland

CDMTCS-290
October 2006

Centre for Discrete Mathematics and
Theoretical Computer Science

On T-codes and necklaces

T. Aaron Gulliver
Department of Electrical and Computer Engineering
University of Victoria, P.O. Box 3055, STN CSC, Victoria, BC, V8W 3P6 Canada
a.gulliver@ieee.org

Ulrich Speidel
Department of Computer Science
The University of Auckland, Private Bag 92019, Auckland, New Zealand
ulrich@cs.auckland.ac.nz

October 25, 2006

Abstract

This paper describes a newly discovered aspect of T-codes, a family of variable length codes first described by Titchener in the 1980s. We show that the constant-length subsets of codewords generated during the T-code construction process constitute sets of necklaces. Moreover, we show that these sets are complete except for a small subset which we characterize. Bounds on the number of constant-length subsets of codewords are determined based on necklaces.

1 Necklaces

Strings of length ℓ over an alphabet A may be divided into equivalence classes under a cyclic shift. That is, two strings $x = a_1a_2 \dots a_\ell$ and $y = b_1b_2 \dots b_\ell$ over A are equivalent under cyclic shift if and only if there is an integer $i \leq \ell$ such that:

$$a_ia_{i+1} \dots a_\ell \dots a_1a_2 \dots a_{i-1} = b_1b_2 \dots b_\ell \quad (1)$$

All strings equivalent to x under under cyclic shift form an *equivalence class*. The lexicographically smallest string in the set is typically known as a *necklace* [9]. The set of all necklaces of length ℓ thus gives rise to the set of all strings of length ℓ if all cyclic shifts of each necklace are included. For simplicity, we shall deviate slightly from convention in this paper and call a set of strings a *set of necklaces* whenever no two distinct strings from the set are equivalent

under cyclic shift. That is, we will not insist that the representative string of an equivalence class be its lexicographically smallest member. Moreover, we will use the term *necklace* to refer to all strings within the same equivalence class.

Example: The six necklaces of length 4 are given by 0000, 0001, 0011, 0101, 0111, and 1111 in accordance with conventional notation. The strings 0001, 0110, and 1010 constitute a set (not complete) of necklaces under the convention adopted here, whereas the strings 0000, 0001, 0010, and 0101 do not.

A necklace that can be formed by concatenation of $k \geq 1$ copies of a string x is said to have *period* x with *repetition factor* k. Necklaces that do not have a repetition factor larger than 1 are called *Lyndon words* [9]. They also represent the irreducible polynomials over $GF(\#A)$.

The number of necklaces of length ℓ is given by

$$\#N_\ell = \frac{1}{\ell} \sum_{d|\ell} \phi(\ell/d) \#A^d \tag{2}$$

where $\phi(m)$ is Euler's totient function, the number of numbers less than m and relatively prime to m. The number of Lyndon words of length ℓ is given by

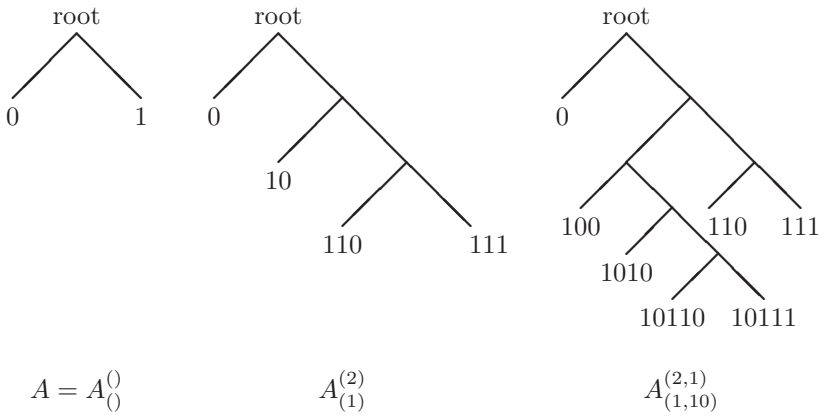
$$\#L_\ell = \frac{1}{\ell} \sum_{i|\ell} \mu(i) \#A^{\ell/i} \tag{3}$$

where $\mu(m)$ is the Mobius function, which is 0 if m is not the product of distinct primes, $+1$ if it is the product of an even number of distinct primes, and -1 otherwise, with $\mu(1) = 1$.

2 T-codes

T-codes [2, 3, 6] are variable prefix codes similar to Huffman codes [1]. Every T-code is *complete*, a property also called *exhaustive* by some authors, i.e., all internal nodes in its decoding tree are fully populated. *Prefix codes* are also sometimes known as *prefix-free* codes, with either terminology implying that no codeword in the set is a proper prefix of another. Similarities largely end there, however. Most Huffman codes are not T-codes, and from a modern perspective, the significance of T-codes is not in Huffman-like source coding. Rather than letting the source symbol probabilities determine the length of the codewords and the structure of the decoding tree, T-codes are constructed without regard to symbol probabilities. Their construction focuses instead on a recursive tree structure.

Every finite alphabet is a (trivial) T-code set by default, with the letters being primitive codewords, and a primitive decoding tree consisting of one internal



.

Figure 1: T-augmentation as a copy-and-append process of decoding trees

node (the root) and $\#A$ leaf nodes. Further, T-code sets may then be constructed via an iterative process called *T-augmentation*. Each T-augmentation involves picking a codeword from the existing T-code set, which is then called the *T-prefix* for that step, and a positive integer called the *T-expansion parameter* or *copy factor* [7]. Any T-code set may thus be derived from an alphabet in a series of n successive T-augmentations using a series of T-prefixes $p_1, p_2, \ldots, p_n$ and a series of T-expansion parameters $k_1, k_2, \ldots, k_n$. The resulting set is denoted $A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ and is said to be a T-code set at *T-augmentation level* n.

The T-augmentation itself is performed according to the following equation:

$$A^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})} = \bigcup_{i=0}^{k_{n+1}} \{p^i_{n+1} s | s \in A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)} \backslash \{p_{n+1}\}\} \cup \{p^{k+1}_{n+1}\} \qquad (4)$$

where $A^{()}_{()} = A$.

In the tree context, this corresponds to making k_i new copies of the existing tree at each step and linking them via the leaf node p_i. Figure 1 depicts the construction of a T-code set $S^{(2,1)}_{(1,10)}$. In the first T-augmentation in this example, the tree for S is copied three times and the three copies are linked to each other via the respective leaf nodes corresponding to the codeword 1. The second T-augmentation links two copies of the tree for $A^{(2)}_{(1)}$ via the leaf node 10.

By T-augmenting repeatedly, we can generate sets of arbitrary size.

2.1 T-code self-synchronization

As many of the proofs in this paper rely on arguments involving T-code self-synchronization, we briefly review its principles here in order to aid understanding.

Using codewords from a T-code set, we may generate arbitrarily long messages by concatenation of codewords from the set. We may observe that each codeword in $A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ is a message composed from codewords in $A^{(k_1,k_2,\ldots,k_i)}_{(p_1,p_2,\ldots,p_i)}$ for all $i < n$. We may speak of a *codeword hierarchy* in this case. Note further that a codeword in $A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ is always composed of between 0 and k_n successive copies of p_n, followed by an arbitrary codeword from $A^{(k_1,k_2,\ldots,k_{n-1})}_{(p_1,p_2,\ldots,p_{n-1})}$.

A T-code decoder for $A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ is a state machine that parses an input string over A into codewords from $A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$. If the input string is allowed to start at an arbitrary point within a codeword from $A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$, the decoder may – at least initially – produce erroneous output. A T-code decoder may be said to produce erroneous output if the start and end positions of codewords identified by the decoder do not correspond to actual codeword boundaries in the string. In this case, the decoder is said to be *unsynchronized*. As further input is read by the decoder, it may eventually identify the correct boundaries. The decoder is said to be *fully synchronized* if and only if the decoder can deduce by observation of the parsed portion of the input string that it is parsing the string into correct codewords at correct positions within the string.

T-codes are known to be highly self-synchronizing, and their self-synchronization mechanism is well understood [6]. That is, a T-code decoder for $A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ will fully synchronize in any semi-infinite sequence that does not start with the pattern:

$$\zeta = p_1^{r_1} p_2^{r_2} \ldots p_n^{r_n} \tag{5}$$

where $0 \leq r_i \leq \infty$ and $\sum_{i=1}^{n} r_i = \infty$. Note that this implies that ζ must contain an infinite substring of the form p_i^{∞} for some $i \leq n$ in order to prevent the decoder from synchronizing fully, with $p_i \neq p_j^{k_j+1}$ for all $j < i$. In this case, the decoder synchronizes fully up to the lower-level T-code set $A^{(k_1,k_2,\ldots,k_{i-1})}_{(p_1,p_2,\ldots,p_{i-1})}$.

The rationale behind these patterns is the following: Consider a decoder decoding a message composed of codewords in $A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$. Let the decoder start in an unsynchronized state at an arbitrary symbol within the message. At this point, the decoder is only synchronized with respect to A, and it is said to be at *synchronization level* 0. That is, the decoder is able to ascertain the end of the present symbol from A and the beginning of the next, but cannot tell where within the string any of the codewords from the higher level T-code sets start or end.

As the decoder reads symbols from the message, note that any symbol other than the first level T-prefix p_1 marks the end of codewords in $A_{p_1}^{k_1}$. This is a direct consequence of the T-augmentation construction. Hence, once one of these symbols is received, the decoder can deduce that it has found the end of a codeword in $A_{p_1}^{k_1}$. It is now synchronized to this set and is said to be at synchronization level 1.

As the decoder achieves synchronization to level $i-1$, it has just read the last symbol of a codeword in $A_{(p_1,p_2,\ldots,p_{i-1})}^{(k_1,k_2,\ldots,k_{i-1})}$ and knows that the following symbols are the start of a codeword in $A_{(p_1,p_2,\ldots,p_{i-1})}^{(k_1,k_2,\ldots,k_{i-1})}$. Unless this codeword is p_i, the decoder has achieved synchronization to level i after decoding it. If the codeword is p_i, the situation is ambiguous: The decoder may be at the end of a codeword $A_{(p_1,p_2,\ldots,p_i)}^{(k_1,k_2,\ldots,k_i)}$, but because these codewords can also be a concatenation starting with p_i, the decoder may only be part-way through a codeword in $A_{(p_1,p_2,\ldots,p_i)}^{(k_1,k_2,\ldots,k_i)}$. To resolve the situation, the decoder will have to read the next codeword at level $i-1$, and so on, until it encounters a codeword that is not p_i. Hence, an infinite sequence of p_i prevents a T-code decoder from synchronizing fully.

Example: Consider $A_{(1,10)}^{(2,1)}$ as in Figure 1. Consider a decoder synchronizing a concatenation of the codewords 110, 1010, and 111. The subscripts indicate the synchronization level before/after each symbol is read (from the left to the right):

$$_0 1_0 1_0 0_1 10_1 10_1 11$$

Note that here, synchronization to level 1 is initially delayed by two copies of $p_1 = 1$, subsequently synchronization to level 2 is held up while $p_2 = 10$ is received. Once 0 from $A_{(1)}^{(2)}$ is encountered at level 1, it is obvious that the decoder has reached the end of a codeword in $A_{(1,10)}^{(2,1)}$, as only the codewords 0 and 100 in $A_{(1,10)}^{(2,1)}$ end with $0 \in A_{(1)}^{(2)}$, and it is not equal to $p_2 = 10$.

2.2 Systematic T-augmentation

For the purposes of this paper, we will restrict ourselves to the discussion of T-codes for which all $k_i = 1$. These are known as the *simple T-codes*. Moreover, we will restrict ourselves to the discussion of T-codes generated using *systematic T-augmentation*, which generates a special class of simple T-codes. In systematic T-augmentation, each p_i is chosen from one of the shortest codewords in the set such that $A_{(p_1,p_2,\ldots,p_{i-1})}^{(k_1,k_2,\ldots,k_{i-1})}$ contains no codeword that is shorter than $|p_i|$. Equation (4) thus simplifies to:

$$A_{(p_1,p_2,\ldots,p_{n+1})} = \{p_{n+1}s | s \in A_{(p_1,p_2,\ldots,p_n)}\} \cup A_{(p_1,p_2,\ldots,p_n)} \backslash \{p_{n+1}\}. \quad (6)$$

We may observe two facts about this type of T-augmentation: Firstly, all

new codewords generated by each systematic T-augmentation are at least twice as long as the T-prefix p_i used in the T-augmentation, because all other codewords in $A_{(p_1,p_2,\ldots,p_n)}$ are at least of length $|p_i|$. Secondly, a codewords that is generated remains in $A_{(p_1,p_2,\ldots,p_n)}$ and its T-augmented sets until it is used as a T-prefix in a T-augmentation.

3 T-code codewords generated during systematic T-augmentation

While T-codes are variable-length codes, subsets of codewords of length ℓ are observed for all ℓ in the course of an infinitely iterating systematic T-augmentation. The subset of codewords of length ℓ exists in the respective T-code sets in its entirety

- from the T-augmentation in which the last available T-prefix $p_i \leq \ell/2$ was used
- until the T-augmentation in which the first codeword of length ℓ is used as the T-prefix.

Table 3 lists the codewords generated in the process for a binary alphabet, the T-augmentation level at which they first appear in the set, and their lexicographically smallest equivalent under cyclic shift. Note that each codeword of length ℓ is created by prefixing a T-prefix of at most length $\ell/2$ to a suffix of at least length $\ell/2$. The point of concatenation is indicated by a dot for each codeword. The astute reader may have noticed that for each length the strings represent a set of necklaces, and that the set of necklaces is complete for $\ell = 2$ and $\ell = 4$. The proof for the necklace property will be given later.

4 Codeword groups of equal length are sets of necklaces

Based on the evidence presented above, we may state the following theorem:

Theorem 4.1 *Consider the T-code set $A_{(p_1,p_2,\ldots,p_n)}$, which has been derived from A through systematic T-augmentation. Any group of codewords of length ℓ from $A_{(p_1,p_2,\ldots,p_n)}$ constitutes a set of necklaces, i.e., no codeword in the group is a cyclic shift of another codeword.*

Proof: Assume that two codewords x, y exist in $A_{(p_1,p_2,\ldots,p_n)}$ with $x \neq y$, $|x| = |y| = \ell$ such that x is a cyclic shift of y. Now consider a semi-infinite string z

Level	Codeword	Shifted
0	0	0
0	1	1
1	0.0	00
1	0.1	01
2	1.1	11
2	1.00	001
2	1.01	101
3	00.00	0000
3	00.01	0001
3	00.11	0011
4	01.01	0101
4	01.11	0111
5	11.11	1111
3	00.100	00001
3	00.101	00101
4	01.100	00011
4	01.101	01011
5	11.100	00111
5	11.101	01111
4	01.0000	000001
4	01.0001	000101
4	01.0011	001101
5	11.0000	000011
5	11.0001	000111
5	11.0011	001111
5	11.0101	010111
5	11.0111	011111
6	100.100	001001
6	100.101	001011
7	101.101	011011
...	...	...

Table 1: Codewords up to length 6 created under systematic T-augmentation.

constructed by concatenating an infinite number of copies of x. The string z will thus also contain a semi-infinite concatenation of y, starting $i < \ell$ positions from the beginning of the sequence. Further, consider a T-code decoder for $A_{(p_1,p_2,\ldots,p_n)}$, which starts decoding at an arbitrary point in z. This decoder will eventually either fully synchronize or will encounter a substring p_i^∞ of ζ and remain synchronized to level $i-1$ only.

- If the decoder fully synchronizes, we have a contradiction because the decoder would have to synchronize with respect to both the series of x and the series of y. However, it can only output either a series of x or a series of y. Hence, x and y cannot both be codewords in $A_{(p_1,p_2,\ldots,p_n)}$.
- If the decoder encounters a substring p_i^∞ and thus does not synchronize fully, then both x and y must necessarily be of the form p_i^k, i.e., $x = y$.

Therefore a cyclic shift of a codeword x can only be another codeword y if $x = y$. QED.

We have now shown that T-code codewords of length ℓ forms sets of necklaces. However, an inspection of the experimental data reveal that the sets of necklaces generated by systematic T-augmentation are not complete. For example, the necklaces 000 and 111 are missing for $\ell = 3$. The next section provides an identification and construction of these "missing" necklaces.

5 The "missing" necklaces

It is also interesting to look at which necklaces do *not* get created during the systematic T-augmentation process, and why. The only way to create 000 and 111, for example, would have been to concatenate a 0 with a 00 or a 1 with a 11. Both options are infeasible due to the disappearance of the required T-prefix along with the appearance of the suffix. This is a general rule, which we may formulate as follows:

Theorem 5.1 *Let x be a codeword in a T-code set $A_{(p_1,p_2,\ldots,p_n)}$ generated under systematic T-augmentation. Systematic T-augmentation only generates codewords of the form x^k as a concatenation of x with k being a power of 2. x^k is never generated if k has at least one odd factor greater than 1.*

To facilitate the proof of this theorem, we first prove the following lemma:

Lemma 5.2 *Let x be a codeword in a T-code set $A_{(p_1,p_2,\ldots,p_n)}$ generated under systematic T-augmentation. Then systematic T-augmentation will eventually generate all codewords of the form x^{2^q} for $q = 1, 2, 3 \ldots$.*

We can prove this lemma by induction: x exists and using $x^{2^{q-1}}$ as a T-prefix generates x^{2^q} as a codeword. QED.

The proof of the theorem then follows from the construction algorithm: By Lemma 5.2, x^k is generated whenever k is a power of 2.

Now assume that k has an odd factor and denote by κ the smallest power of 2 larger than k. Then we know that codewords of length $|x^\kappa|$ are generated by using T-prefixes of length $|x^{\kappa/2}|$ or smaller. This includes x^κ being generated by prefixing $x^{\kappa/2}$ to itself. Since there are no codewords shorter than $x^{\kappa/2}$ in the set at the time that x^κ is generated, codewords shorter than $|x^\kappa|$ can no longer be generated. If x^k is generated, it must be generated before $x^{\kappa/2}$ is used as a T-prefix. However, as x^k is longer than $x^{\kappa/2}$, it would not have been used as a T-prefix yet, so it would still have to be in the T-code set at that time. As we know that $x^{\kappa/2}$ is in the set and $x^{\kappa/2}$ is a prefix of x^k, the existence of x^k would violate the prefix-free property of the set. Hence, x^k is never generated. QED.

Example: At $\ell = 6$, ℓ has the odd factor 3, with codewords of the form x^3 not being constructed. With three codewords of length 2 being available as choices for x, namely 00, 01, and 11, the strings 000000, 010101 and 111111 never appear as codewords, even though they are all necklaces of length 6. Consequently, there are only 11 rather than 14 sequences of length 6 present.

Note also that when multiple odd factors are present, care must be taken to account for all non-generated necklaces. In the case of $\ell = 30$, we have factors 2, 3, and 5, but the odd divisors we need to consider are 3, 5, and 15. The strings of length 30 that are not created as codewords include:

1. codewords of length 1 concatenated 30 times:0^{30} and 1^{30}.
2. codewords of length 2 concatenated 15 times: $(00)^{15}$, $(01)^{15}$, and $(11)^{15}$, but note that the first and last of these are identical with 0^{30} and 1^{30}, so one does not need to take the first item into account.
3. codewords of length 3 concatenated 10 times: $(100)^{10}$ and $(101)^{1}0$.
4. codewords of length 5 concatenated 6 times: $(00100)^6$, $(00101)^6$, $(01100)^6$, $(01101)^6$, $(11100)^6$ and $(11101)^6$.
5. codewords of length 6 concatenated 5 times. Note that these include the codewords in the third item as both 100100 and 101101 are generated, so the codewords under this item do not need to be taken into account separately.
6. codewords of length 10 concatenated 3 times. These include all codewords of codewords of length 5 concatenated 6 times, so the fourth item is surplus to requirements.

This leaves us with the non-generation of codewords of the form x^{15} with $|x| = 2$, x^5 with $|x| = 6$ and x^3 with $|x| = 10$.

Theorem 5.1 gives rise to another useful theorem:

Theorem 5.3 *Each codeword y generated by systematic T-augmentation can be written as x^k such that k is a power of two and x is a codeword in some T-code set $A_{(p_1,p_2,\ldots,p_n)}$ generated in the course of the same systematic T-augmentation sequence. Iff $k = 1 = 2^0$ is the only solution for k, y is a Lyndon word.*

Proof: The proof is similar to that of Theorem 4.1. If y is a Lyndon word, it follows immediately from the definition of Lyndon words that $k = 1 = 2^0$ is the only solution for k. It remains to be shown that if y is not a Lyndon word, there must be a solution for $k > 1$ with k being a power of 2. If y is not a Lyndon word, then there must be a string x and some $k > 1$ such that $y = x^k$. By inspection of the code sets, the theorem holds for small ℓ. Now consider a semi-infinite sequence formed by semi-infinite concatenation of y. A T-code decoder cannot fully synchronize because the location of the actual codeword boundaries is ambiguous as a result of the factor being greater than one. We know, however, that synchronization is prevented by strings of the form p_i^∞. Hence, y must be a concatenation of codewords of the form p_i^2. Since systematic T-augmentation creates only concatenations of the form $(p_i^2)^k$ where k is a power of 2, the theorem holds. QED.

The "missing necklaces" – or, as we are restricted to claiming at the moment, some of them – can then be constructed by concatenating codewords x of length $|x| = \ell/k$ into strings x^k for odd divisors $k > 1$ of ℓ. Note that this construction inherently implies that these necklaces cannot be Lyndon words as they have periods $\ell/k < \ell$.

We next prove that these "missing necklaces" actually constitute a set of necklaces. Denote by M_ℓ the set of all strings $y = x^k$ with $|y| = \ell$, k odd, and $x \in A_{(p_1,p_2,\ldots,p_n)}$ for some n under systematic T-augmentation.

Lemma 5.4 *M_ℓ forms a set of necklaces.*

Proof: By Theorem 5.3, the number of repetitions of a period in a codeword x generated by systematic T-augmentation is always a power of two (including possibly 2^0). Thus, the periods in two strings constructed with different k repeat a different number of times, so the two strings cannot represent the same necklace. Moreover, two different strings for a given k cannot represent the same necklace either, because the codewords they are based on represent different necklaces. QED.

We now show that M_ℓ does not overlap with the necklaces generated by systematic T-augmentation.

Lemma 5.5 *M_ℓ and the set of necklaces of length ℓ generated by systematic T-augmentation are disjoint.*

Proof: All necklaces in M_ℓ have periods shorter than ℓ, and the number of times they repeat in each of these necklaces is divisible by an odd number larger than 1. All necklaces in the second set have periods that repeat a power of two times. This rules out any overlap. QED.

Denote by m_ℓ the maximum number of codewords of length ℓ generated during systematic T-augmentation. Then

$$\#M_\ell = \sum_{k|\ell, k \text{ odd}} m_k. \tag{7}$$

6 Completeness

The question now arises as to whether the union of the set of codewords of length ℓ and M_ℓ is a complete set of necklaces. If it is complete, then it is not possible to add an additional string of length ℓ that represents a new equivalence class.

The number of necklaces of length ℓ is given by (2). m_ℓ is given by (25) in [8] as

$$m_\ell = \frac{\#A^\ell}{\ell} - \sum_{k|\ell, k<\ell} (-1)^{\frac{\ell}{k}+1} \frac{k m_k}{\ell} \tag{8}$$

If our assertion holds, then the following theorem must be true.

Theorem 6.1 *The union of the set of all codewords of length ℓ generated under systematic T-augmentation and the corresponding set M_ℓ constitute the complete set of necklaces N_ℓ and*

$$\#N_\ell = m_\ell + \#M_\ell, \tag{9}$$

i.e.,

$$\sum_{k|\ell} \phi(\ell/k) \#A^k = \#A^\ell - \sum_{k|\ell, k<\ell} (-1)^{\frac{\ell}{k}+1} k m_k + \ell \sum_{k|\ell, k \text{ odd}} m_k \tag{10}$$

holds for all ℓ under systematic T-augmentation.

Proof: We have already shown that the two constituent sets are disjoint. It is thus a necessary and sufficient condition to show that any necklace of length ℓ has to belong to one of these sets. Consider $A_{(p_1,p_2,\ldots,p_n)}$ such that $|p_n| = \ell - 1$ and $A_{(p_1,p_2,\ldots,p_n)}$ contains no further codewords of length $\ell - 1$. Assume that there exists a necklace y of length ℓ that is neither in $A_{(p_1,p_2,\ldots,p_n)}$ nor in M_ℓ. By inspection, we can decide whether y has a period of less than ℓ or not. The argument is then as follows:

- If y has a period of length $\ell/k < \ell$, we may use induction over ℓ. We know from inspection that the theorem holds for small ℓ. The string x with $x^k = y$ must thus have a shift equivalent x' that is a codeword of length ℓ/k or is in $M_{\ell/k}$. If k has an odd factor or $x' \in M_{\ell/k}$, y has a shift equivalent y' that is in M_ℓ. If k is a power of 2 and x' was a codeword, y has a shift equivalent y' that is in the current T-code set. Thus, y, if it exists, must be aperiodic.

- If y is aperiodic, consider a semi-infinite string y_∞ of copies of y and a T-code decoder for $A_{(p_1,p_2,\ldots,p_n)}$ synchronizing into this string. Since y is aperiodic and $|y| > |p_n|$, the y_∞ cannot contain patterns of the form p_i^∞. The decoder must therefore synchronize fully. Since our assumption states that y and its cyclic shifts are not in $A_{(p_1,p_2,\ldots,p_n)}$, y_∞ cannot contain codewords of length ℓ from $A_{(p_1,p_2,\ldots,p_n)}$, and so a synchronized decoder cannot encounter them. Instead, it must now be decoding codewords larger than ℓ as the set does not contain codewords shorter than ℓ. Now consider two decoders starting at two positions in y_∞, with the first position arbitrary and the second separated by ℓ symbols from the first. After a finite synchronisation delay, both decoders are fully synchronized and output the same sequence of codewords at the same codeword boundaries. As the synchronization process is invariant under a shift of ℓ symbols (both decoders are fed the same data), the second decoder achieves synchronization exactly ℓ symbols after the first. Since the points of synchronization are codeword boundaries at level n, the string between the boundaries is a codeword of length ℓ, which we have just concluded must not exist in y_∞.

Theorem 6.1 thus holds. QED.

We can now state the following results.

Lemma 6.2 *Codewords x generated by systematic T-augmentation are Lyndon words where $|x|$ is an odd prime. If $|x|$ is a power of 2, all necklaces of length $|x|$ are generated.*

Proof: Sequences of prime lengths are always aperiodic or have period 1, and the period 1 sequences are not generated as codewords because the number of repetitions required is odd. For $|x| = 2^q$, the length has no odd factor, so $M_{2^q} = \emptyset$. QED.

The following two corollaries provide bounds on m_ℓ.

Corollary 6.3 *The number of necklaces of length ℓ is an upper bound for m_ℓ.*

Proof: This follows immediately from Theorem 6.1. QED.

Corollary 6.4 *The number of Lyndon words of length ℓ is a lower bound for m_ℓ.*

Proof: This follows from Theorem 6.1 and the fact that M_ℓ does not contain Lyndon words. QED.

7 Conclusions

A correspondence between T-code codewords and necklaces has been established. The systematic T-code augmentation process generates all necklaces except for a specific subset. This subset has been characterized. These results were used to derive a new expression for the number of codewords which is much simpler than that in [8]. The number of necklaces provides an upperbound on the number of codewords, while the number of Lyndon words provides a lower bound.

References

[1] D. Huffman: *A Method for the Construction of Minimum Redundancy Codes*. Proc. Inst. Radio Eng., 40:1098-1101, September 1952

[2] M. R. Titchener: *Generalized T-codes: an Extended Construction Algorithm for Self-Synchronizing Variable-Length Codes*, IEE Proceedings – Computers and Digital Techniques, 143(3), June 1996, pp. 122-128.

[3] U. Guenther: *Data Compression and Serial Communication with Generalized T-Codes*, Journal of Universal Computer Science, V. 2, N 11, 1996, pp. 769-795. `http://www.iicm.edu/jucs_2_11`

[4] U. Guenther, P. Hertling, R. Nicolescu, and M. R. Titchener: *Representing Variable-Length Codes in Fixed-Length T-Depletion Format in Encoders and Decoders*, CDMTCS Research Report no.44, Centre of Discrete Mathematics and Theoretical Computer Science, The University of Auckland, August 1997. `http://www.cs.auckland.ac.nz/CDMTCS/researchreports/044ulrich.pdf`.

[5] U. Guenther, P. Hertling, R. Nicolescu, and M. R. Titchener: *Representing Variable-Length Codes in Fixed-Length T-Depletion Format in Encoders and Decoders*, Journal of Universal Computer Science, 3(11), Nov. 1997, pp. 1207–1225. `http://www.iicm.edu/jucs_3_11`.

[6] U. Guenther: *Robust Source Coding with Generalized T-Codes*. PhD Thesis, The University of Auckland, 1998. `http://www.tcs.auckland.ac.nz/~ulrich/phd.ps.gz`

[7] W. Ebeling, R. Steuer, and M. R. Titchener: *Partition-Based Entropies of Deterministic and Stochastic Maps*, *Stochastics and Dynamics*, 1(1), p. 45., March 2001.

[8] M. R. Titchener and A. Gulliver and R. Nicolescu and U. Speidel and L. Staiger: *Deterministic Complexity and Entropy*, *Fundamenta Informaticae*, v. 64(1-4), 443-461, 2005

[9] K. Cattell, F. Ruskey, J. Sawada, M. Serra and C.R. Miers, *Fast Algorithms to Generate Necklaces, Unlabeled Necklaces, and Irreducible Polynomials over GF(2)*, *J. Algorithms*, vol. 37, pp. 267–282, 2000.