SOME GENERALIZATIONS ON COUNTING BINARY STRINGS

JOSHUA T. ABBOTT
NEW COLLEGE OF FLORIDA

Abstract. Extending R. Grimaldi's work on binary strings and Jacobsthal numbers for the language $A = \{0, 01, 11\}$, we will examine some general properties for counting binary languages. We focus mainly on counts for the number of strings of length n inside the Kleene closure of a given language. We will discuss how these counts are affected when adding additional elements to a language. We also present counts for the number of 0's and 1's inside these binary strings of length n.

Key Phrases: binary strings, Jacobsthal numbers, symbol codes

1. Introduction - Trivial Generalizations of a_n

Starting with the alphabet $\Sigma = \{0, 1\}$, let A be the language $\{0, 01, 11\}$, a subset of Σ^*, the Kleene closure of Σ, as in Grimaldi[1]. Let a_n count the number of distinct binary strings of length n in A^*, the Kleene closure of A. Then $a_n = a_{n-1} + 2a_{n-2}$, for all $n \geq 3$, with a_1=1, a_2=3; because to obtain a string of length n, we either append 0 to the right of a string of length *n-1* in A^*, or we append 01 or 11 to the right of a string of length *n-2* in A^*. Since the initial conditions can be determined easily, we could solve this recurrence relation to obtain $a_n = J_n$, the n^{th} Jacobsthal number. The rest of this paper will only be concerned with finding recurrence relations in general for binary languages.

Let A^c be the bitwise complement of a language A. For example, if $A = \{0, 01, 11\}$ as before, then $A^c = \{1, 10, 00\}$, and by an analogous argument as given before, $a^c{}_n = a^c{}_{n-1} + 2a^c{}_{n-2}$. Thus, A and A^c share the same a_n, the number of binary strings of length n. It is also rather trivial to note if $\Sigma \subseteq A$, then $a_n = 2^n$.

One last simple general result is if $A = \{\ x_1,\ x_2,\ \dots\ ,\ x_m\ \}$ with $|x_1| = |x_2| = \dots = |x_m| = \beta$, then

$$a_n = \begin{cases} m^{\frac{n}{\beta}} & n(mod\beta) \equiv 0 \\ 0 & else \end{cases}$$

2. Some Non-trivial Generalizations of a_n

Definition 2.1. A code C over an alphabet Σ is *uniquely decipherable* if whenever $c_1,...,c_k$, $d_1,...,d_j$ are codewords in C and

$$c_1 \ldots c_k = d_1 \ldots d_j$$

then $k = j$, and $c_i = d_i$, for all $i = 1,...,k$.

A code is analogous to a language, with codewords being the elements of the language. We will use the terms code and language interchangeably.

Proposition 2.1. *If A is a language with k_i elements of length i, then*

$$a_n \leq \sum k_i a_{n-i}$$

with equality iff A is uniquely decipherable.

Proof. The summation $\sum k_i a_{n-i}$ is simply a count of appending strings of length k_i to the previous strings of length a_{n-i}. If the language A is uniquely decipherable, then every string in A^* can be decomposed in exactly one way; thus $a_n = \sum k_i a_{n-i}$ in this case. Otherwise, if A is not uniquely decipherable, there are multiple decompositions of strings in A^*, thus a_n is strictly less than $\sum k_i a_{n-i}$. $\square$

Example 2.1. Let $A = \{0, 01, 11\}$ as in Grimaldi[1]. Then $k_1 = 1$, $k_2 = 2$, and $a_n \leq (1)a_{n-1} + (2)a_{n-2}$. Further, A is uniquely decipherable, so $a_n = a_{n-1} + 2a_{n-2}$.

Theorem 2.1 (McMillan's Theorem)**.** *If $C = \{\ c_1,\ c_2,\ \ldots\ ,\ c_q\ \}$ is a uniquely decipherable binary code with l_i = length(c_i), then its codeword lengths l_1, l_2, ..., l_q must satisfy Kraft's inequality:*

$$\sum_{k=1}^{q} \frac{1}{2^{l_k}} \leq 1$$

Proof. Refer to [2] for a standard proof of this theorem. $\square$

Example 2.2. Let $A = \{0, 01, 11\}$ as in Grimaldi[1]. Then $l_1 = 1$, $l_2 = 2$, $l_3 = 2$, $q = 3$, and $\displaystyle\sum_{k=1}^{q} \frac{1}{2^{l_k}} = \frac{1}{2} + \frac{1}{4} + \frac{1}{4} = 1$.
Since the summation equals 1, adding more elements to A violates Kraft's inequality and will result in a language that is not uniquely decipherable, and thus the count a_n' for this resulting language will be strictly less than $\sum k_i a_{n-i}'$.

There exist methods to compute a_n if A is not uniquely decipherable. We present the following three cases.

Case 1 (Linear Dependence): We can find a linearly independent set from the elements of the language A by removing the linearly dependent elements to yield language $A^{'}$ with an equivalent count $a_n \equiv a_n^{'}$.

Example 2.3. $A = \{0, 00\}$ can be reduced to $A^{'} = \{0\}$, and thus $a_n = a_{n-1}$, with $a_1 = 1$, clearly.

Example 2.4. $A = \{0, 10, 11, 001110\}$ can be reduced to $A^{'} = \{0, 10, 11\}$, since $001110 = 2(0) + (10) + (11)$. $A^{'}$ is uniquely decipherable, thus
$a_n = a_{n-1} + 2a_{n-2}$.

Case 2 (Symmetry): If there exist elements $(x)(y)$ and $(y)(x)$ in A, with either x or y also also an element in A, then there is a function $f(n)$ to account for the strings in A^* that have multiple decompositions. Thus $a_n = \sum k_i a_{n-i} - f(n)$.

Example 2.5. Let $A = \{0, 01, 10\}$. We see $010 \in A^*$ decomposes as both $(0)(10)$ and $(01)(0)$. The resulting $f(n) = a_{n-3}$ and thus $a_n = a_{n-1} + 2a_{n-2} - a_{n-3}$.

Case 3 (Least Common Multiple): If there exist elements x,y in A such that $z \in A^*$ can be decomposed as $(x)(y)$ or $(y)(x)$, or a string $z^{'} \in A^*$ that can be decomposed as either i copies of (x) or j copies of (y), then there is a function *g(n)* to account for the strings in A^* that have multiple decompositions. Thus $a_n = \sum k_i a_{n-i} - g(n)$.

Example 2.6. Let $A = \{1, 00, 000\}$. We see $00000 \in A^*$ decomposes as both $(00)(000)$ and $(000)(00)$, and $000000 \in A^*$ decomposes as both $(00)(00)(00)$ and $(000)(000)$. As a result, $g(n) = a_{n-3} - a_{n-4}$ and thus $a_n = a_{n-1} + a_{n-2} + a_{n-3} - (a_{n-3} - a_{n-4}) = a_{n-1} + a_{n-2} + a_{n-4}$.

The methods used in these cases or a combination of them result in an exact computation for a_n.

3. Zeros and Ones in the a_n Binary Strings

Let z_n and w_n count the number of zeros and ones respectively that occur among the a_n binary strings of length n in A^*.

Example 3.1. Let $A = \{0, 01, 11\}$ as in Grimaldi[1]. Then $z_n = (z_{n-1} + a_{n-1}) + (z_{n-2} + a_{n-2}) + (z_{n-2})$ where $(z_{n-1} + a_{n-1})$ takes the previous count of zeros and sums this with the number of elements in a_{n-1} since we are adding one 0 to each, $(z_{n-2} + a_{n-2})$ takes the penultimate count of zeros and sums this with the number of elements in a_{n-2} since we are adding a two bit string 01 which has just one 0, and (z_{n-2}) accounts for adding the two bit string 11 to every element in a_{n-2}, but since there are no new zeros in 11, we simply use the previous zero count z_{n-2}.
$w_n = (w_{n-1}) + (w_{n-2} + a_{n-2}) + (w_{n-2} + 2a_{n-2})$ is constructed from a similar argument as given above.

It is clear that $w_n + z_n = na_n$ and thus $w_n + z_n \leq n \sum k_i a_{n-i}$. There is an interesting underlying structure to these counts that may reveal methods of finding exact counts for w_n and z_n.

Proposition 3.1. *Let m_j count the number of occurrences of 0 in strings of length j in A, and let k_i count the number of elements of length i in A. Then*

$$z_n \leq \sum(k_i z_{n-i}) + \sum(m_j a_{n-j})$$

with equality iff A is uniquely decipherable.

In lieu of a proof, consider a deeper discussion of the previous example (3.1). As previously shown, $z_n = (z_{n-1} + a_{n-1}) + (z_{n-2} + a_{n-2}) + (z_{n-2})$. Rearranging the terms, we get $z_n = (z_{n-1} + 2z_{n-2}) + (a_{n-1}) + (a_{n-2})$. Similarly, as previously shown, $w_n = (w_{n-1}) + (w_{n-2} + a_{n-2}) + (w_{n-2} + 2a_{n-2})$, and rearranging terms we get $w_n = (w_{n-1} + 2w_{n-2}) + (3a_{n-2})$. Recall that $a_n = a_{n-1} + 2a_{n-2}$ for this language. So w_n and z_n can be constructed from the count a_n and terms that account for the number of zeros and ones respectively in the a_{n-i} strings. A formal proof is analogous to counting argument given for Proposition 2.1.

We have not developed methods to derive exact counts for w_n and z_n, but conjecture they are similar to the methods used in the three cases given for computing a_n.

4. Conclusion

We have reviewed methods to compute a_n for arbitrary binary languages and also a bound for counting the number of zeros and ones in these a_n strings. The examples given should help clarify the efficacy of the methods and motivate deeper investigation.

5. References

1.) Grimaldi, R., Binary Strings and the Jacobsthal Numbers. *Congressus Numerantium*, Volume 174, 2005, Pp. 3-22.

2.) Roman, S., *Coding and Information Theory.* Published by Springer, 1992.

E-mail address: `joshua.abbott@ncf.edu`