Feedback with Carry Shift Registers and Bent Sequences

MIDN 1/C Charles Celerier

May 2, 2012

Abstract

A stream cipher uses pseudorandom sequences to mimic the security of a one-time pad. This paper will investigate how Bent functions can be used to generate Bent sequences with large 2-adic span for use in feedback with carry shift registers (FCSRs). The linear complexity of an FCSR, a commonly used device in stream ciphers, directly relates to the magnitude of the 2-adic span of the sequence generated. This affects the resistance of the stream cipher against register synthesis attacks. In this paper, it is shown that it is possible to compute the 2-adic value of a Bent sequence.

1 Introduction

Bytes of data, or short sequences of 1s and 0s, are exchanged between computer systems each day on public channels. Because gentlemen do read each other's mail, it is necessary to secure the communication of private data sent over public channels. The solution to this problem is solved by *cryptography*, the designing of systems to secure data exchanges over public channels.

The following example is the classical communication scenario presented in many books on cryptography. Let there be two parties, Alice and Bob, who wish to communicate with one another. A third party, Eve, is a potential eavesdropper. Alice wants to send a message, known as the *plaintext*, to Bob. To accomplish this without Eve knowing what the message is before it is received by Bob, Alice must *encrypt* her message by some prearranged method, usually involving an *encryption key*, to generate a related message called *ciphertext*. The idea is that the sent ciphertext, even if it is intercepted by Eve, will be too difficult to interpret and will conceal the plaintext message. Upon receipt of the ciphertext, Bob will *decrypt* the message, usually involving a *decryption key*, similar to the encryption of the message, and obtain the plaintext message. A visual of this scenario is presented in 1.

This scenario is the standard example found in many different introductory cryptography references. Cryptographers have created numerous encryption and decryption schemes, or *cryptosystems*, to secure the messages sent between Alice and Bob. Many of these systems have been broken because of the amount

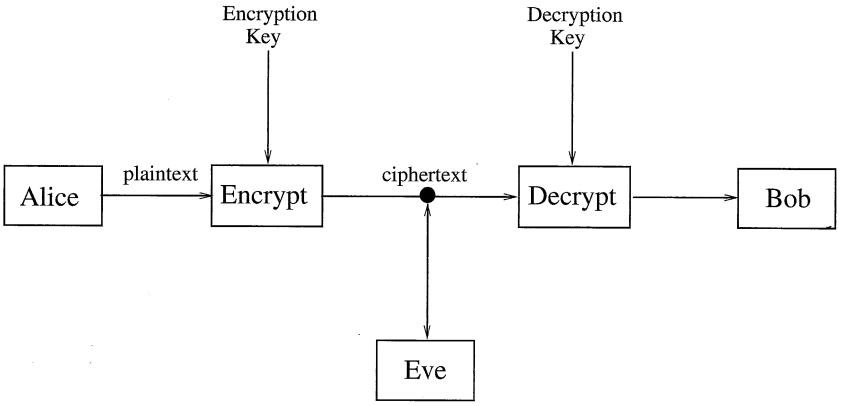

Figure 1: The Basic Comunication Scenario for Cryptography [22]

of work that goes into the study of breaking cryptosystems, called *cryptanalysis*. A constant battle exists between the designers and breakers of cryptosystems, strengthening designs and attacks every day. In fact, designing a strong cryptosystem typically requires knowledge of cryptography and cryptanalysis.

Before diving into the topic of this paper, there is an important assumption that must be presented. When designing a cryptosystem, every cryptographer assumes *Kerckhoffs' principle* [22]: "In assessing the security of a cryptosystem, one should always assume the enemy knows the method being used." The security of a cryptosystem cannot be based on the concealment of the encryption and decryption algorithms. In practice, the enemy can obtain the algorithms in many ways, including the defection or capture of people. The security must be based solely on the key.

Imagine that Alice's message to Bob is a sequence of 1s and 0s, as it would be in the real world. Consider a cryptosystem which encrypts each bit in the sequence separately. This would be done by what is called a *stream cipher*, which, according to Rueppel, divides bit sequences into individual bits and enciphers each bit with a time-varying function whose time-dependency is governed by the internal state of the stream cipher [19]. The stream cipher can also be thought of in terms of a *keystream* which is a sequence of 1s and 0s the same length of the message that is added to the message using addition in $\mathbb{F}_2$ (also known as XOR). If the keystream was perfectly random, then the cryptosystem would be unbreakable, or *perfectly secret*, as discovered by Claude Shannon in his famous paper "Communication Theory of Secrecy Systems," written in 1945 and published in 1949. This cryptosystem is known as the *one-time pad*. Though it is perfectly secret, it can be difficult to implement because of the inability to produce perfectly random keystreams. Constructing a method to produce perfectly random sequences is a contradiction in itself.

Though it is impossible to create perfectly random sequences, it is possible

to get close. This type of sequence is called a *pseudorandom sequence*. These sequences have extremely long periods but are statistically indistinguishable from random sequences. This paper will discuss the construction of pseudorandom sequences using a particular finite state machine called the *feedback with carry shift register*. It will also investigate the possibility of incorporating *Bent functions*, or perfectly non-linear Boolean functions, into the construction of psuedorandom sequences in an attempt to resist synthesizing algorithms.

2 Boolean Functions

This section will establish the definition of a *Boolean function* and how to write these functions as polynomials. First, the finite field with two elements, denoted $\mathbb{F}_2$, is defined. The definitions and notations will follow those found in [4].

2.1 Review of Boolean Functions

The two element field $(\mathbb{F}_2, \oplus, \cdot)$ is the set $\{0, 1\}$ with defined binary operations $\oplus$ and $\cdot$, also commonly referred to as the *XOR* and *AND* operators, respectively.

XOR	AND
$0 \oplus 0 := 0$	$0 \cdot 0 := 0$
$0 \oplus 1 := 1$	$0 \cdot 1 := 0$
$1 \oplus 0 := 1$	$1 \cdot 0 := 0$
$1 \oplus 1 := 0$	$1 \cdot 1 := 1$

Table 1: Binary Operations for $\mathbb{F}_2$

It should be clear that $(\mathbb{F}_2, \oplus, \cdot)$ is a commutative ring with an identity. Additionally, the only non-zero element 1 is it's own inverse. Therefore $(\mathbb{F}_2, \oplus, \cdot)$ is a finite field, which will now be denoted by $\mathbb{F}_2$. The n-dimensional vector space over $\mathbb{F}_2$ will be denoted by $\mathbb{F}_2^n$, with the usual inner product. Components of these vectors, the individual 1s and 0s, will be known as *bits*. When a bit is *flipped* this means it changed from a 1 to a 0, or vice versa. For two vectors $x, y \in \mathbb{F}_2^n$ where $x = (x_0, \cdots, x_{n-1})$ and $y = (y_0, \cdots, y_{n-1})$, the inner product in $\mathbb{F}_2^n$ will be defined as $x \cdot y := x_0 \cdot y_0 \oplus \cdots \oplus x_{n-1} \cdot y_{n-1}$.

Example 2.1.1. Let $a, b \in \mathbb{F}_2^3$ such that $a = (1, 0, 1)$ and $b = (0, 1, 1)$ then

$$a + b = (1 \oplus 0, 0 \oplus 1, 1 \oplus 1) = (1, 1, 0)$$
$$a \cdot b = 1 \cdot 0 \oplus 0 \cdot 1 \oplus 1 \cdot 1 = 1$$

Each vector in $\mathbb{F}_2^n$ can be uniquely represented by an integer between 0 and $2^n - 1$. To do this, the components of each vector in $\mathbb{F}_2^n$ are trivially mapped to the integers 0 and 1, and then used in the one-to-one binary representation

function B:

$$B : \mathbb{F}_2^n \to \mathbb{N} \cup \{0\} \text{ such that } B(u) := \sum_{i=0}^{n-1} u_i \cdot 2^i. \tag{1}$$

This definition of B has created the convention where the *least significant bit* appears on the left and the *most significant bit* appears on the right. The *Hamming weight* and *Hamming distance* functions are used to count the number of 1s in a vector and count the number of differences between two vectors in $\mathbb{F}_2^n$. These are fundmental functions in coding theory and are useful when talking about Boolean functions.

Definition 2.1.2. Let $x, y \in \mathbb{F}_2^n$. Then $wt : \mathbb{F}_2^n \to \mathbb{N} \cup \{0\}$ is defined by

$$wt(x) := \sum_{i=0}^{n-1} x_i$$

and $d : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{N} \cup \{0\}$ is defined by

$$d(x, y) := wt(x + y).$$

Then $wt(x)$ is the *Hamming weight* of x and $d(x, y)$ is the *Hamming distance* between x and y.

Remark. The Hamming weight of a vector $x \in \mathbb{F}_2^n$ is the number of 1s in the vector, and the Hamming distance between two vectors $x, y \in \mathbb{F}_2^n$ is the number of bit differences between the two vectors.

Definition 2.1.3. Let $x \in \mathbb{F}_2^n$. Then $supp : \mathbb{F}_2^n \to \mathcal{P}(\mathbb{N} \cup \{0\})$ is defined by

$$supp(x) := \{i \in \mathbb{N} \cup \{0\} : x_i = 1\}$$

Example 2.1.4. Let $a, b, c \in \mathbb{F}_2^5$ such that

$$a = (0, 1, 1, 0, 1),\ b = (1, 1, 1, 0, 0),\ \text{and } c = (0, 0, 1, 1, 0).$$

Then,

$$\begin{array}{lll} wt(a) = 3 & supp(a) = \{1, 2, 4\} & d(a, b) = 2 \\ wt(b) = 3 & supp(b) = \{0, 1, 2\} & d(a, c) = 3 \\ wt(c) = 2 & supp(c) = \{2, 3\} & d(b, c) = 3. \end{array}$$

If we define the vectors $v_i \in \mathbb{F}_2^n$ such that $v_i = B^{-1}(i)$ for $0 \leq i \leq 2^{n-1}$, then the sequence $(v_0, v_1, \ldots, v_{2^n-1})$ is said to be in *lexicographical order*. Using the convention of lexicographical ordering and the inner product in $\mathbb{F}_2^n$, a $2^n \times 2^n$ matrix can be written which captures all of the possible inner products of two vectors in $\mathbb{F}_2^n$. This is a matrix where $x_i \cdot x_j$ appears in the ith row and jth column. This particular matrix turns out to be a *Hadamard matrix* which will be studied later.

There is an interesting orthogonality property in the vector space $\mathbb{F}_2^n$ known as the *orthogonality principle* that every non-zero vector in $\mathbb{F}_2^n$ is orthogonal to exactly half of the vectors in the vectorspace.

Theorem 2.1.5. *Let $x \in \mathbb{F}_2^n$. Then*

$$\sum_{y \in \mathbb{F}_2^n} (-1)^{x \cdot y} = 2^n \text{ for } x = 0$$
$$= 0 \text{ otherwise.}$$

Proof. Let $x = 0 \in \mathbb{F}_2^n$. Then $\forall y \in \mathbb{F}_2^n$, $x \cdot y = 0$, so $(-1)^{x \cdot y} = 1$. Therefore, $\sum_{y \in \mathbb{F}_2^n} (-1)^{x \cdot y} = |\mathbb{F}_2^n| = 2^n$.

Let $x \in \mathbb{F}_2^n$ where $x \neq 0$. Assume the ith bit of x is non-zero and define $e_i \in \mathbb{F}_2^n$ as a vector with all zero bits except for the ith bit which is 1. Then

$$\begin{aligned} \sum_{y \in \mathbb{F}_2^n} (-1)^{x \cdot y} &= \sum_{y \in \mathbb{F}_2^n} (-1)^{x \cdot (y + e_i)} \\ &= \sum_{y \in \mathbb{F}_2^n} (-1)^{x \cdot y} (-1)^{x \cdot e_i} \\ &= - \sum_{y \in \mathbb{F}_2^n} (-1)^{x \cdot y}. \end{aligned}$$

Therefore, $\sum_{y \in \mathbb{F}_2^n} (-1)^{x \cdot y} = - \sum_{y \in \mathbb{F}_2^n} (-1)^{x \cdot y}$, which implies $\sum_{y \in \mathbb{F}_2^n} (-1)^{x \cdot y} = 0$. □

When introduced to a new vector space, it is natural to begin looking at functions in that field. The particular function of interest here will be what is known as a *Boolean function*.

Definition 2.1.6. Any function f defined such that

$$f : \mathbb{F}_2^n \to \mathbb{F}_2$$

is a *Boolean function*. The set of all Boolean functions on n variables will be denoted by $\mathcal{BF}_n$.

The number of Boolean functions increases extremely rapidly as the number of variables increases.[1]

$$|\mathcal{BF}_n| = 2^{2^n} \tag{2}$$

The Boolean function f is presented in a *truth table* in Table 2. The Hamming weight of f is the number of 1s that f has when evaluated at every point in the $\mathbb{F}_2^n$:

$$wt(f) = |\{u \in \mathbb{F}_2^n : f(u) = 1\}|.$$

[1] With today's fastest supercomputer operating at 10.51 petaflops (the K computer in Japan), if one floating point operation was expended visiting every Boolean function of 7 variables, it would take over a thousand trillion years to complete the process. This length of time is roughly 70,000 times the age of the universe. Though every symmetric cryptosystem in use can be broken down into several Boolean functions of several variables, it would be infeasible to brute force search through all of the possibilities of Boolean functions which reconstruct the cryptosystem.

x_0	x_1	x_2	x_3	$f(x_0, x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	0	0
0	0	1	0	1
1	0	1	0	0
0	1	1	0	1
1	1	1	0	0
0	0	0	1	0
1	0	0	1	0
0	1	0	1	1
1	1	0	1	0
0	0	1	1	0
1	0	1	1	0
0	1	1	1	1
1	1	1	1	1

Table 2: Truth Table of f

2.2 Boolean Polynomials

A truth table is not a very compact method to define a Boolean function. It is much more efficient and easier to implement a Boolean function when written as a formula. This can be done by writing a Boolean function algebraically. As a first step toward defining f algebraically a one-to-one and onto function will be defined which maps every f in $\mathcal{BF}_n$ to a vector in $\mathbb{F}_2^{2^n}$. This will be the function $V : \mathcal{BF}_n \to \mathbb{F}_2^{2^n}$ such that

$$V(f) := (f(v_0), \ldots, f(v_{2^n-1})) \text{ where } v_i = B^{-1}(i). \tag{3}$$

It is trivial to show that addition is homomorphic under V,

$$V(f_1 \oplus f_2) = V(f_1) \oplus V(f_2).$$

Then the standard basis of $\mathbb{F}_2^{2^n}$ can be used to pull back to an equivalent basis of $\mathcal{BF}_n$. Let $e_i \in \mathbb{F}_2^{2^n}$ be defined so that

$$\begin{aligned} e_0 &= (1, 0, \ldots) \\ e_1 &= (0, 1, \ldots) \\ &\vdots \\ e_{2^n-1} &= (0, \ldots, 0, 1). \end{aligned}$$

The *atomic Boolean functions* will be defined as the $f_i \in \mathcal{BF}_n$ where there exists an $e_i \in \mathbb{F}_2^{2^n}$ such that $V(f_i) = e_i$. Every vector of $\mathbb{F}_2^{2^n}$ can be written as a linear combination of the standard basis vectors, and equivalently, every

Boolean function is a linear combination of atomic Boolean functions. This means for every $u \in \mathbb{F}_2^{2^n}$ there exists a set of $c_i \in \mathbb{F}_2$ such that

$$\begin{aligned} u &= c_0 e_0 \oplus \cdots \oplus c_{2^n-1} e_{2^n-1} \\ \Leftrightarrow f &= c_0 f_0 \oplus \cdots \oplus c_{2^n-1} f_{2^n-1}. \end{aligned}$$

where $V(f) = u$.

The function f defined in Table 2 can be written as a linear combination of the atomic Boolean functions in $\mathcal{BF}_4$. Since the coefficients in the linear combinations are either 0 or 1, it is true that every Boolean function can be written as the sum of $wt(f)$ atomic Boolean functions.

x_0	x_1	x_2	x_3	f	f_1	f_2	f_4	f_6	f_{10}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	0
0	1	0	0	1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	1	0	0
1	1	0	1	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	1	0
1	1	1	1	1	0	0	0	0	0	0	1

Table 3: f broken into atomic Boolean function in $\mathcal{BF}_4$

The equation $f = f_1 + f_2 + f_4 + f_6 + f_{10} + f_{14} + f_{15}$ should be clear from Table 3. Knowing how to write the atomic Boolean functions as polynomials would lead to knowing how to write any Boolean function as a polynomial. The polynomials representing the Boolean functions will belong to the the polynomial ring $\mathbb{F}_2[x_0, \cdots, x_{n-1}]/(x_0^2 \oplus x_0, \cdots, x_{n-1}^2 \oplus x_{n-1})$. To properly represent an atomic Boolean function, a polynomial must equal 1 at only one vector $(x_0, \cdots, x_{n-1})$. Recall the support function from Definition 2.1.3. Then the polynomial respresenting each Boolean function is as follows:

$$f_i = \Bigg(\prod_{j \in supp(B^{-1}(i))} x_j \Bigg) \Bigg(\prod_{j \notin supp(B^{-1}(i))} (1 \oplus x_j) \Bigg). \tag{4}$$

Proof. Let $x = B^{-1}(i)$ where $x = (x_0, \cdots, x_{n-1})$. Then $\{x_i : x_i = 1\} = \{x_i :$

$i \in supp(x)\}$. Therefore,

$$f_i(x) = \Bigg(\prod_{j \in supp(B^{-1}(i))} x_j \Bigg) \Bigg(\prod_{j \notin supp(B^{-1}(i))} (1 \oplus x_j) \Bigg) = 1.$$

Let $x \neq B^{-1}(i)$. Then,

$$\Bigg(\prod_{j \in supp(B^{-1}(i))} x_j \Bigg) = 0$$

$$\therefore f_i(x) = 0.$$

□

Now the function f from Table 2 can be written as the sum of the following atomic polynomials:

$$\begin{aligned}
f_1 &= (1 \oplus x_3)(1 \oplus x_2)(1 \oplus x_1)x_0 \\
&= x_0 \oplus x_1x_0 \oplus x_2x_0 \oplus x_2x_1x_0 \oplus x_3x_0 \oplus x_3x_1x_0 \oplus x_3x_2x_0 \oplus x_3x_2x_1x_0 \\
f_2 &= (1 \oplus x_3)(1 \oplus x_2)x_1(1 \oplus x_0) \\
f_4 &= (1 \oplus x_3)x_2(1 \oplus x_1)(1 \oplus x_0) \\
f_6 &= (1 \oplus x_3)x_2x_1(1 \oplus x_0) \\
f_{10} &= x_3(1 \oplus x_2)x_1(1 \oplus x_0) \\
f_{14} &= x_3x_2x_1(1 \oplus x_0) \\
f_{15} &= x_3x_2x_1x_0
\end{aligned}$$

After summing the atomic polynomials upon multiplying them out,

$$f = x_0x_1x_2x_3 \oplus x_0x_1x_3 \oplus x_0x_3 \oplus x_0 \oplus x_1x_2x_3 \oplus x_1x_2 \oplus x_1 \oplus x_2x_3 \oplus x_2.$$

Now the uniqueness of the polynomial representation for each Boolean function is considered. This is easily seen by considering the uniqueness of each Boolean function and the size of the polynomial ring.

Theorem 2.2.1. *Each n-variable Boolean function is uniquely represeneted as a polynomial in the polynomial ring $\mathbb{F}_2[x_0, \cdots, x_{n-1}]/(x_0^2 \oplus x_0, \cdots, x_{n-1}^2 \oplus x_{n-1})$. Let $f \in \mathcal{BF}_n$. Then there exists a unique set of $a_I \in \mathbb{F}_2$ such that*

$$f(x) = \sum_{I \in 2^{\{0,\ldots,n-1\}}} a_I \Bigg(\prod_{i \in I} x_i \Bigg) \tag{5}$$

Proof.

$$\begin{aligned}
|\mathcal{BF}_n| &= |\mathbb{F}_2^{2^n}| \\
&= |\{(a_\emptyset, \ldots, a_I, \ldots) : a_I \in \mathbb{F}_2\}| \\
&= |\mathbb{F}_2[x_0, \cdots, x_{n-1}]/(x_0^2 \oplus x_0, \cdots, x_{n-1}^2 \oplus x_{n-1})|
\end{aligned}$$

Because every Boolean function is determined by at least one polynomial and the size of the polynomial ring equals the size of the set of all Boolean functions, each Boolean function must be uniquely determined by a polynomial in the polynomial ring. □

2.3 The Walsh Transform

The Walsh Transform is similar to the Discrete Fourier Transform. This transformation has many applications in signals analysis. The idea is to express a given signal as a function of frequency to reveal the dominant frequencies in the signal. For this paper, the Walsh Transform is considered over the Discrete Fourier Transform because this is the transformation used by Rothaus in his original definition of Bent functions published in the Journal of Combinatorial Theory in 1976 [20].

It is necessary to discuss *characters* as a preliminary to Walsh Transforms. The definitions in this section follow [18].

Definition 2.3.1. A *character* χ of a finite abelian group G is a group homomorphism from G into the multiplicative group of complex numbers.

For the purposes of this paper, it should be clear that $\chi_\lambda(x) := (-1)^{\lambda \cdot x}$ where $\lambda, x \in \mathbb{F}_2^n$ is a *group character* of $\mathbb{F}_2^n$. Define the *dual group* $\hat{\mathbb{F}_2^n}$ to be the group of all characters of $\mathbb{F}_2^n$. The group operation in $\hat{\mathbb{F}_2^n}$ is pointwise multiplication of functions:

$$(\chi \cdot \psi)(x) = \chi(x)\psi(x),\ x \in \mathbb{F}_2^n.$$

This operation is closed under multiplication.

Lemma 2.3.2. $\mathbb{F}_2^n \cong \hat{\mathbb{F}_2^n}$

Proof. Let $\Upsilon : \mathbb{F}_2^n \to \hat{\mathbb{F}_2^n}$ where $\Upsilon(\lambda) := \chi_\lambda$. Every character in $\hat{\mathbb{F}_2^n}$ corresponds to an element of $\mathbb{F}_2^n$. Thus, $|\mathbb{F}_2^n| = |\hat{\mathbb{F}_2^n}|$. Therefore if Υ is one-to-one, then it must be an isomorphism.
Let $\Upsilon(\lambda_1) = \Upsilon(\lambda_2)$. Then $\forall x$

$$\begin{aligned}(-1)^{\lambda_1 \cdot x} &= (-1)^{\lambda_2 \cdot x} \\ &= (-1)^{(\lambda_1 + \lambda_1 + \lambda_2) \cdot x} \\ &= (-1)^{\lambda_1 \cdot x}(-1)^{(\lambda_1 + \lambda_2) \cdot x}.\end{aligned}$$

Finally, $(\lambda_1 + \lambda_2) \cdot x = 0$ for all $x \in \mathbb{F}_2^n$, which implies $\lambda_1 + \lambda_2 = 0$. Therefore, $\lambda_1 = \lambda_2$. □

Addition in $\mathbb{F}_2^n$ corresponds to multiplication in $\hat{\mathbb{F}_2^n}$. By definition

$$(\chi_{\lambda_1} \cdot \chi_{\lambda_2})(x) = \chi_{\lambda_1}(x)\chi_{\lambda_2}(x) = (-1)^{\lambda_1 \cdot x}(-1)^{\lambda_2 \cdot x} = (-1)^{(\lambda_1 + \lambda_2) \cdot x} = \chi_{\lambda_1 + \lambda_2}(x).$$

Definition 2.3.3. Let $f \in \mathcal{BF}_n$. Then $\hat{f} : \mathbb{F}_2^n \to \{1, -1\}$ such that $\hat{f}(x) = (-1)^{f(x)}$ is a *pseudo-Boolean function*

Lemma 2.3.4. *The characters of $\mathbb{F}_2^n$ belong to $\hat{\mathcal{BF}}_n = \{\hat{f} : f \in \mathcal{BF}_n\}$ and form an orthonormal basis of $\hat{\mathcal{BF}}_n \otimes \mathbb{R}$.*

Proof. The dimension of $\mathcal{B}\hat{\mathcal{F}}_n$ is 2^n, and there are 2^n characters of $\mathbb{F}_2^n$.

$$\sum_{x\in\mathbb{F}_2^n} \chi_{\lambda_i}(x)\cdot\chi_{\lambda_j}(x) = \sum_{x\in\mathbb{F}_2^n} (-1)^{(\lambda_i+\lambda_j)\cdot x}$$
$$= \begin{cases} 0 & \text{if } i \neq j \\ 2^n & \text{if } i = j. \end{cases}$$

Therefore, the characters of $\mathbb{F}_2^n$ form an orthonormal basis of $\hat{\mathcal{BF}}_n \otimes \mathbb{R}$. □

Definition 2.3.5. Let $f \in \mathcal{BF}_n$ and $\lambda \in \mathbb{F}_2^n$. Then the *Walsh transform* of f is defined by:

$$\mathcal{W}_f(\lambda) = \sum_{x\in\mathbb{F}_2^n} \hat{f}(x)\chi_\lambda(x). \tag{6}$$

Every pseudo-Boolean function can be written as a linear combination of the characters of $\mathbb{F}_2^n$. The coefficients in these linear combinations reveal important properties of the functions. Rothaus rewrote the pseudo-Boolean function as a linear combination of characters as follows [20].

Lemma 2.3.6. *For $\hat{f} \in \hat{\mathcal{BF}}_n$,*

$$\hat{f}(x) = \frac{1}{2^{n/2}} \sum_{\lambda\in\mathbb{F}_2^n} c(\lambda)\chi_\lambda(x) \tag{7}$$

where $c(\lambda)$ are given by

$$c(\lambda) = \frac{1}{2^{n/2}}\mathcal{W}_f(\lambda) \tag{8}$$

Each $c(\lambda)$ is called a *Fourier coefficient* of f. As observed by Rothaus in [20], $2^{n/2}c(\lambda)$ is the number of zeros minus the number of ones of the function $f(x)+\lambda\cdot x$. The Hamming weight of f is easily determined using the zero Fourier coefficient $c(0)$:

$$c(0) = \frac{1}{2^{n/2}} \sum_{x\in\mathbb{F}_2^n} (-1)^{f(x)}$$
$$= \frac{1}{2^{n/2}}\big((2^n - wt(f)) - wt(f)\big)$$

$$\Rightarrow wt(f) = 2^{n-1} - 2^{n/2-1}c(0). \tag{9}$$

Clearly $\mathcal{W}_f(\lambda) = \frac{1}{2^{n/2}}c(\lambda)$. For large $|\mathcal{W}_f(\lambda)|$, the Hamming distance between f and an affine function in $\mathcal{BF}_n$ is small.

Example 2.3.7. There are a two cases that should be clear to the reader.

1. Let $f(x) = \lambda \cdot x$. Then $\mathcal{W}_f(\lambda) = 2^n$.
2. Let $f(x) \neq \lambda \cdot x \ \ \forall x$. Then $f(x) = \lambda \cdot x + 1 \ \ \forall x$. So, $\mathcal{W}_f(\lambda) = -2^n$

In both cases, f is an affine function.

2.4 Bent Functions

These functions are useful in cryptographic applications because they add resistance to differential attacks as a result of being *perfectly nonlinear*. As mentioned before, these were defined by Rothaus in 1976. Then definition of a bent function is presented here.

Definition 2.4.1. If all of the Fourier coefficients of $\hat{f}$ are ± 1 then f is a *bent function*.

Theorem 2.4.2. [20] *If f is a bent function on $\mathbb{F}_2^n$, then n is even, $n = 2k$. Moreover, the degree of f is at most k, except in the case $k = 1$.*

Proof. $c(\lambda) = \pm 1$. This implies $2^{n/2}c(\lambda)$ is an integer. Therefore n must be even.

Let $n = 2k$ with $k > 1$, and let $r > k$. Consider the polynomial $f(x, 0, 0, \ldots, 0) = g(x)$ where $x = (x_1, x_2, \ldots, x_r)$ (up to this point all indexing has started at 0; it is more convenient in this proof to begin numbering at 1). Then by Equation (7),

$$\hat{g}(x) = \frac{1}{2^{r/2}} \sum_{\lambda_1, \lambda_2, \ldots, \lambda_r = 0,1} b(\lambda_1, \ldots, \lambda_r)\chi_{(\lambda_1, \ldots, \lambda_r)}(x)$$

and

$$\hat{f}(x, 0) = \frac{1}{2^{n/2}} \sum_{\lambda_1, \lambda_2, \ldots, \lambda_n = 0,1} c(\lambda_1, \ldots, \lambda_n)\chi_{(\lambda_1, \ldots, \lambda_n)}(x, 0).$$

Because $f(x, 0) = g(x)$ and the uniqueness of the Fourier expansion, b and c are related such that

$$b(\lambda_1, \ldots, \lambda_r) = \frac{1}{2^{(n-r)/2}} \sum_{\lambda_{r+1}, \ldots, \lambda_n = 0,1} c(\lambda_1, \ldots, \lambda_r, \lambda_{r+1}, \ldots, \lambda_n).$$

Then,

$$\begin{aligned} wt(f(x, 0)) &= wt(g(x)) \\ &= 2^{r-1} - 2^{r/2-1}b(0) \\ &= 2^{r-1} - 2^{r-n/2-1} \sum_{\lambda_{r+1}, \ldots, \lambda_n = 0,1} c(0, \ldots, 0, \lambda_{r+1}, \ldots, \lambda_n). \end{aligned}$$

There are 2^{n-r} summands in $\sum c(0, \ldots, 0, \lambda_{r+1}, \ldots, \lambda_n)$. Since f is bent, $c(\lambda) = \pm 1$. By rewriting $1 = -1 + 2$,

$$\begin{aligned} \sum c(0, \ldots, 0, \lambda_{r+1}, \ldots, \lambda_n) &= -2^{n-r} + 2wt(c(0, \ldots, 0, \lambda_{r+1}, \ldots, \lambda_n)) \\ &= 2\big(wt(c(0, \ldots, 0, \lambda_{r+1}, \ldots, \lambda_n)) - 2^{n-r-1}\big) \end{aligned}$$

Thus, $wt(g(x))$ is even. This implies that $g(x)$ is the sum of an even number of atomic Boolean functions. Therefore the coefficient of $x_1 x_2 \cdots x_r$ in the polynomial representing $g(x)$ must be 0. This is true for every $r > k$, so the degree of f must not be greater than k. □

2.5 Constructions of Bent Functions

A simple bent function construction is accomplished by the Boolean functions in the *Maiorana-McFarland class*. This is the set $\mathcal{M}$ which contains all Boolean function on $\mathbb{F}_2^n = \{(x, y) : x, y \in \mathbb{F}_2^{n/2}\}$, of the form

$$f(x, y) = x \cdot \pi(y) \oplus g(y)$$

where π is any permutation on $\mathbb{F}_2^{n/2}$ and g is any Boolean function on $\mathbb{F}_2^{n/2}$.

All functions in the Maiorana-McFarland class of Boolean functions are bent.

2.6 Monotone Boolean Functions

In the introduction to Boolean functions, each Boolean function was shown as a sum of atomic Boolean functions. This section will show how it is possible to construct every Boolean function from monotone Boolean functions.

Define a partial order $\leq$ on $\mathbb{F}_2^n$ as follows: for each $v, w \in \mathbb{F}_2^n$,

$$v \leq w$$

whenever $supp(v) \subseteq supp(w)$. (This partial order is not to be confused with the lexicographical order.) A Boolean function is called *monotone* if whenever we have $v \leq w$ then we also have $f(v) \leq f(w)$. In the case $supp(v) \subset supp(w)$, $v < w$ may be written.

Each monotone function corresponds to a unique set of vectors which determine where the function begins mapping to 1s.

Definition 2.6.1. Let f be an n-variable monotone Boolean function. Every $x \in \mathbb{F}_2^n$ where for all $x' \in \mathbb{F}_2^n$ such that $x' < x$, $f(x') < f(x)$ is called a *least support* of f.

Definition 2.6.2. If an n-variable monotone Boolean function f has exactly one least support, then f is an *atomic monotone Boolean function.*

Theorem 2.6.3. *All monomials are monotone Boolean functions.*

Proof. Let $f(x) = \prod_{i \in I} x_i$ for $I \in 2^{\{0,\cdots,n-1\}}$ and $y \in \mathbb{F}_2^n$. Then $supp(y) \supseteq I$ if and only if $f(y) = 1$ and $supp(y) \not\supseteq I$ if and only if $f(y) = 0$. Let $u, v \in \mathbb{F}_2^n$ such that $supp(u) \subseteq supp(v)$ and $f(u) = 1$. Then, $I \subseteq supp(u) \subseteq supp(v)$. This implies $f(v) = 1$. Thus, f is a monotone Boolean function. □

Example 2.6.4. In the following examples, the two functions $f = x_0x_2$ and $g = x_1x_2$ are monomials. Each function is also a monotone function. For f, every $x \in \mathbb{F}_2^4$ such that $supp(x) \supseteq supp((1, 0, 1, 0))$ implies $f(x) = 1$. For g, $supp(x) \supseteq supp((0, 1, 1, 0))$ implies $g(x) = 1$.

```
sage: B=BooleanPolynomialRing(4,'x')
sage: x0,x1,x2,x3=B.gens()
sage: f=BooleanFunction(x0*x2)
```

```
sage: print_truth(f)
(0, 0, 0, 0) 0 False
(1, 0, 0, 0) 0 False
(0, 1, 0, 0) 0 False
(1, 1, 0, 0) 0 False
(0, 0, 1, 0) 0 False
(1, 0, 1, 0) 1 True
(0, 1, 1, 0) 0 False
(1, 1, 1, 0) 0 True
(0, 0, 0, 1) 0 False
(1, 0, 0, 1) 0 False
(0, 1, 0, 1) 0 False
(1, 1, 0, 1) 0 False
(0, 0, 1, 1) 0 False
(1, 0, 1, 1) 0 True
(0, 1, 1, 1) 0 False
(1, 1, 1, 1) 0 True
```

```
sage: g=BooleanFunction(x1*x2)
sage: print_truth(g)
(0, 0, 0, 0) 0 False
(1, 0, 0, 0) 0 False
(0, 1, 0, 0) 0 False
(1, 1, 0, 0) 0 False
(0, 0, 1, 0) 0 False
(1, 0, 1, 0) 0 False
(0, 1, 1, 0) 1 True
(1, 1, 1, 0) 0 True
(0, 0, 0, 1) 0 False
(1, 0, 0, 1) 0 False
(0, 1, 0, 1) 0 False
(1, 1, 0, 1) 0 False
(0, 0, 1, 1) 0 False
(1, 0, 1, 1) 0 False
(0, 1, 1, 1) 0 True
(1, 1, 1, 1) 0 True
```

Every monotone Boolean function can be written as a sum of monomials based on the least supports of the function. This is done by unioning in every way all of the least supports. If Γ is the set of all the least supports of a monotone Boolean function, then this unioning is defined by the function S:

$$S: \ 2^{\Gamma} \to 2^{\{0,\cdots,n-1\}} \tag{10}$$

$$S(\alpha) := \bigcup_{v \in \alpha} supp(v) \tag{11}$$

Theorem 2.6.5. *Let f be an n-variable monotone Boolean function with least supports Γ. Then*

$$f(x) = \sum_{I \in S(2^\Gamma),\ I \neq \emptyset} x^I \tag{12}$$

Proof. Define $g(x) = \sum_{I \in S(2^\Gamma),\ I \neq \emptyset} x^I$. Pick $y \in \mathbb{F}_2^n$. Let $\alpha_y = \{v \in \Gamma | supp(v) \subseteq supp(y)\}$. It follows that $\beta \subseteq \alpha_y$ and $\beta \neq \emptyset$ if and only if $y^{S(\beta) \cup C} = 1$, $C \subseteq \{0, \cdots, n-1\}$. Let $\beta \subseteq \alpha$. Then $\alpha \neq \emptyset$, therefore $|2^\alpha|$ is even. Thus, the number of β's is odd. In other words, $|\{\beta \subseteq \Gamma$: $y^{S(\beta) \cup C} = 1$ for $C \subseteq \{0, \cdots, n-1\}$ and $C \notin S(2^\Gamma)\}|$ is odd.

Case 1: Suppose $supp(y) \in S(2^\Gamma) \cup C$. Then the number of non-zero terms in $g(y)$ is odd, therfore $g(y) = 1$.

Case 2: Suppose $supp(y) \notin S(2^\Gamma) \cup C$. Then there do not exist any non-zero terms in $g(y)$, therefore $g(y) = 0$.

Then g is 1 at precisely all of the palces f is 1. Therefore $g = f$. □

3 N-adic Integers

The goal of this section is to introduce the reader to the N-adic integer ring and define the *N-adic valuation*. The 2-adic valuation will be used to analyze bent sequences.

3.1 N-adic Integer Ring

The notation used in the definition of the N-adic numbers will follow the same notation used by Borevich and Shafarevich in Chapter 1 of *Number Theory* [1]. In this section, the set of N-adic integers is shown to be a commutative ring with an identity.

Definition 3.1.1. Let N be an integer. Then the infinite integer sequence (x_n) determines an *N-adic integer* α, or $(x_n) \to \alpha$, if

$$x_{i+1} \equiv x_i \pmod{N^{i+1}} \quad \forall i \geq 0. \tag{13}$$

Two sequences (x_n) and (x'_n) determine the same N-adic integer if

$$x_i \equiv x'_i \pmod{N^{i+1}} \quad \forall i \geq 0. \tag{14}$$

The *set of all N-adic integers* will be denoted by $\mathbb{Z}_N$.

Each integer x is associated with a N-adic integer, determined by the sequence $(x,\ x,\ \ldots,\ x,\ \ldots)$. These integers will be called *rational integers* in the N-adic integers.

Example 3.1.2. Let $(x_n) \to \alpha \in \mathbb{Z}_3$. Then the first 5 terms of (x_n) may look something like:

$$\begin{aligned}(x_n) &= (\ 1\ ,\ 1+2\cdot 3\ ,\ 1+2\cdot 3+1\cdot 3^2\ ,\\ &\qquad 1+2\cdot 3+1\cdot 3^2\ ,\ 1+2\cdot 3+1\cdot 3^2+1\cdot 3^4\ ,\ \dots)\\ &= (1,7,16,16,97,\dots)\end{aligned}$$

Then equivalent sequences to (x_n) could begin differently for the first few terms:

$$\begin{aligned}(y_n) &= (4,25,16,178,583,\dots)\\ (z_n) &= (-2,-47,232,-308,97,\dots)\end{aligned}$$

The sequences for (y_n) and (z_n) satisfy equation (13) for the first 5 terms, so they could be N-adic integers up to this point. Also, both are equivalent to (x_n) according to the equivalence defined in equation (14).

$$\begin{aligned}1 &\equiv 4 \equiv 2 \pmod 3\\ 7 &\equiv 25 \equiv -47 \pmod{3^2}\\ 16 &\equiv 16 \equiv 232 \pmod{3^3}\\ 16 &\equiv 178 \equiv -308 \pmod{3^4}\\ 97 &\equiv 583 \equiv 97 \pmod{3^5}\end{aligned}$$

Therefore $(x_n), (y_n), (z_n) \to \alpha$.

Because there are infinitely many sequence representations for any N-adic integer, it is useful to define a canonical sequence to be used when writing N-adic integers as sequences.

Definition 3.1.3. For a given N-adic integer α, a given sequence (a_n) with the properties:

i. $(a_n) \to \alpha$

ii. $(a_n) = (a_0,\ a_0 + a_1\cdot N,\ \dots,\ a_0 + \cdots + a_i\cdot N^i,\ \dots) : 0 \le a_i < N \quad \forall i \ge 0$

will be called *canonical*. The sequence $(a_0, a_1, a_2, \cdots, a_i, \cdots)$ is the *digit representation* of α. When $N \le 10$, the digits are usually written adjacent to one another.

Example 3.1.4. In Example 3.1.2, the sequence (x_n) was a canonical sequence that determined the N-adic integer α. A few more examples of canonical sequences determining 7-adic integers are given here:

$\beta = 3164\dots$, then the canonical sequence $(b_n) \to \beta$ is

$$\begin{aligned}(b_n) &= (3,\ 3+1\cdot 7,\ 3+1\cdot 7+6\cdot 7^2,\ 3+1\cdot 7+6\cdot 7^2+4\cdot 7^3,\ \dots)\\ &= (3,10,304,1676,\dots)\end{aligned}$$

$\gamma = 0164\ldots$, then the canonical sequence $(c_n) \to \gamma$ is

$$\begin{aligned}(c_n) &= (0,\ 1\cdot 7,\ 1\cdot 7 + 6\cdot 7^2,\ 1\cdot 7 + 6\cdot 7^2 + 4\cdot 7^3,\ \ldots)\\ &= (0, 7, 301, 1673, \ldots)\end{aligned}$$

$\delta = 5031\ldots$, then the canonical sequence $(d_n) \to \delta$ is

$$\begin{aligned}(d_n) &= (5,\ 5, 5 + 3\cdot 7^2,\ 5 + 3\cdot 7^2 + 1\cdot 7^3,\ \ldots)\\ &= (5, 5, 152, 495, \ldots).\end{aligned}$$

Definition 3.1.5. Addition and multiplication in $\mathbb{Z}_N$ are done term by term. Let $\alpha, \beta \in \mathbb{Z}_N$ and $(x_n) \to \alpha, (y_n) \to \beta$. Then,

$$\begin{aligned}(x_n) + (y_n) &:= (x_0 + y_0, x_1 + y_1, \ldots) \to \alpha + \beta\\ (x_n) \cdot (y_n) &:= (x_0 \cdot y_0, x_1 \cdot y_1, \ldots) \to \alpha \cdot \beta\end{aligned}$$

Define $(0, 0, 0, \ldots) \to 0 \in \mathbb{Z}_N$ and $(1, 1, 1, \ldots) \to 1 \in \mathbb{Z}_N$

Lemma 3.1.6. *For $\alpha \in \mathbb{Z}_N$, $\alpha + 0 = 0 + \alpha = \alpha$ and $1 \cdot \alpha = \alpha \cdot 1 = \alpha$.*

Proof. Let $(x_n) \to \alpha \in \mathbb{Z}_N$.

$$\begin{aligned}(x_n) + (0, 0, \ldots) &= (x_0 + 0, x_1 + 0, \ldots, x_i + 0, \ldots)\\ &= (x_0, \ldots, x_i, \ldots).\\ (0, 0, \ldots) + (x_n) &= (0 + x_0, 0 + x_1, \ldots, 0 + x_i, \ldots)\\ &= (x_0, \ldots, x_i, \ldots).\end{aligned}$$

$(x_n) = (x_n) + (0, 0, \ldots) = (0, 0, \ldots) + (x_n)$ implies $\alpha = \alpha + 0 = 0 + \alpha$. Therefore, the additive identity in $\mathbb{Z}_N$ is 0.

$$\begin{aligned}(x_n) \cdot (1, 1, \ldots) &= (x_0 \cdot 1, x_1 \cdot 1, \ldots, x_i \cdot 1, \ldots)\\ &= (x_0, \ldots, x_i, \ldots).\\ (1, 1, \ldots) \cdot (x_n) &= (1 \cdot x_0, 1 \cdot x_1, \ldots, 1 \cdot x_i, \ldots)\\ &= (x_0, \ldots, x_i, \ldots).\end{aligned}$$

$(x_n) = (x_n) \cdot (1, 1, \ldots) = (1, 1, \ldots) \cdot (x_n)$ implies $\alpha = \alpha \cdot 1 = 1 \cdot \alpha$. Therefore, the multiplicative identity in $\mathbb{Z}_N$ is 1. □

Finally, this section defines a *ring* and proves that $\mathbb{Z}_N$ is a *commutative ring with an identity.*

Definition 3.1.7. A *ring* is a set R with two binary operations defined on it. These are usually called addition denoted by $+$, and multiplication denoted by $\cdot$ or juxtaposition, satisfying the following six axioms:

1. Addition is commutative: $a + b = b + a \ \ \forall a, b \in R$.
2. Addition is associative: $a + (b + c) = (a + b) + c \ \ \forall a, b, c \in R$.
3. There exists an additive identity, denoted by 0, such that $a+0 = a \ \ \forall a \in R$.
4. $\forall a \in R$ there exists an additive inverse, denoted by $-a$, such that $a + (-a) = 0$.
5. Multiplication is associative: $a(bc) = (ab)c \ \forall a, b, c \in R$
6. Multiplication is left and right distributive over addition:

$$a(b + c) = ab + ac$$
$$(b + c)a = ba + ca$$

If it is also true that

7. Multiplication is commutative: $ab = ba \ \ \forall a, b \in R$, then R is a *commutative ring.*

Further if

8. There exists a multiplicative identity denoted by 1 such that $a \cdot 1 = a$ and $1 \cdot a = a \ \ \forall a \in R$, then R is a *ring with an identity.*

If R satisfies all eight properties, then R is a *commutative ring with an identity.*

Theorem 3.1.8. *$\mathbb{Z}_N$ is a commutative ring with an identity.*

Proof. Let $(x_n), (y_n), (z_n)$ determine $\alpha, \beta, \gamma \in \mathbb{Z}_N$ respectively. Then

1. *Commutativity of Addition*

$$\begin{aligned}(x_n) + (y_n) &= (x_0 + y_0, \dots, x_i + y_i, \dots) \\ &= (y_0 + x_0, \dots, y_i + x_i, \dots) \\ &= (y_n) + (x_n).\end{aligned}$$

   $(x_n) + (y_n) \to \alpha + \beta$ and $(x_n) + (y_n) = (y_n) + (x_n) \to \beta + \alpha$. Therefore, by Definition 3.1.1, $\alpha + \beta = \beta + \alpha$.

2. *Associativity of Addition*

$$\begin{aligned}(x_n) + ((y_n) + (z_n)) &= (x_n) + (y_0 + z_0, \dots, y_i + z_i, \dots) \\ &= (x_0 + (y_0 + z_0), \dots, x_i + (y_i + z_i), \dots) \\ &= ((x_0 + y_0) + z_0, \dots, (x_i + y_i) + z_i, \dots) \\ &= (x_0 + y_0, \dots, x_i + y_i, \dots) + (z_n) \\ &= ((x_n) + (y_n)) + (z_n).\end{aligned}$$

   Therefore, $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$.

3. *Existence of the Additive Identity*

   By Lemma 3.1.6, 0 is the additive identity.

4. *Existence of Additive Inverses*

   Define $-(x_n) = (N - x_0, N^2 - x_1, \dots, N^{i+1} - x_i, \dots) \to -\alpha$. Then

$$\begin{aligned}(x_n) + (-(x_n)) &= (x_0 + N - x_0, x_1 + N^2 - x_1, \dots, x_i + N^{i+1} - x_i, \dots) \\ &= (N, N^2, \dots, N^{i+1}, \dots) \\ &\equiv (0, 0, \dots) \\ &= 0.\end{aligned}$$

   Therefore, $\alpha + (-\alpha) = 0$.

5. *Associativity of Multiplication*

$$\begin{aligned}(x_n)((y_n)(z_n)) &= (x_n)(y_0 z_0, \dots, y_i z_i, \dots) \\ &= (x_0(y_0 z_0), \dots, x_i(y_i z_i), \dots) \\ &= ((x_0 y_0) z_0, \dots, (x_i y_i) z_i, \dots) \\ &= (x_0 y_0, \dots, x_i y_i, \dots)(z_n) \\ &= ((x_n)(y_n))(z_n).\end{aligned}$$

   Therefore, $\alpha(\beta\gamma) = (\alpha\beta)\gamma$.

7. *Commutativity of Multiplication*

$$\begin{aligned}(x_n)(y_n) &= (x_0 y_0, \dots, x_i y_i, \dots) \\ &= (y_0 x_0, \dots, y_i x_i, \dots) \\ &= (y_n)(x_n).\end{aligned}$$

   Therefore, $\alpha\beta = \beta\alpha$.

6. *Left and right distributivity of multiplication over addition*

$$\begin{aligned}(x_n)((y_n) + (z_n)) &= (x_n)(y_0 + z_0, \dots, y_i + z_i, \dots) \\ &= (x_0(y_0 + z_0), \dots, x_i(y_i + z_i), \dots) \\ &= (x_0 y_0 + x_0 z_0, \dots, x_i y_i + x_i z_i, \dots) \\ &= (x_n)(y_n) + (x_n)(z_n).\end{aligned}$$

   By commutativity of multiplication,

$$\begin{aligned}((y_n) + (z_n))(x_n) &= (x_n)((y_n) + (z_n)) \\ &= (x_n)(y_n) + (x_n)(z_n) \\ &= (y_n)(x_n) + (z_n)(x_n).\end{aligned}$$

   Therefore, $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$ and $(\beta + \gamma)\alpha = \beta\alpha + \gamma\alpha$.

8. *Existence of a multiplicative identity*

   By Lemma 3.1.6, 1 is the multiplicative identity.

   Properties 1 through 8 from Definition 3.1.7 are satisfied, so $\mathbb{Z}_N$ is a commutative ring with an identity.

□

Theorem 3.1.9. *An N-adic integer α, which is determined by a sequence (x_n), is a unit if and only if x_0 is relatively prime to N.*

Proof. Let α be a unit. Then there is an N-adic integer β such that $\alpha\beta = 1$. If β is determined by the sequence (y_n), then

$$x_i y_i \equiv 1 \pmod{N^{i+1}} \quad \forall i \geq 0. \tag{15}$$

In particular, $x_0 y_0 \equiv 1 \pmod{N}$, hence x_0 is relatively prime to N.

Conversely, let x_0 be relatively prime to N. Then $x_0 \not\equiv 0 \pmod{N}$. From Equation (13)

$$\begin{aligned} x_1 &\equiv x_0 \pmod{N} \\ &\vdots \\ x_{i+1} &\equiv x_i \pmod{N^i}. \end{aligned}$$

Working backward, $x_{i+1} \equiv x_i \equiv \cdots \equiv x_1 \equiv x_0 \pmod{N}$. Thus, x_i is relatively prime to N $\forall i \geq 0$, which implies x_i is relatively prime to N^{i+1}. Consequently, $\forall i \geq 0$ $\exists y_i$ such that $x_i y_i \equiv 1 \pmod{N^{i+1}}$. Since $x_{i+1} \equiv x_i \pmod{N}^i$ and $x_{i+1} y_{i+1} \equiv x_i y_i \pmod{N^i}$. Then, $y_{i+1} \equiv y_i \pmod{N^i}$. Therefore the sequence (y_n) determines some N-adic integer β. Because $x_i y_i \equiv 1 \pmod{N^{i+1}}$ $\forall i \geq 0$, $\alpha\beta = 1$. This means α is a unit. □

From this theorem it follows that a rational integer $a \in \mathbb{Z}_N$ is a unit if and only if a is relatively prime to N. If a is invertible in $\mathbb{Z}_N$, then for any rational integer $b \in \mathbb{Z}_N$, $b/a = a^{-1}b \in \mathbb{Z}_N$.

3.2 N-adic Valuation and Norm

The 2-adic valuation will be used to analyze the Bent sequences presented later. The general definitions for the N-adic valuation and norm are given here.

Definition 3.2.1. Let $\alpha = (a_n) \in \mathbb{Z}_N \setminus (0)$. If m is the smallest number in $\mathbb{N}$ such that $a_m \not\equiv 0 \pmod{N^{m+1}}$, then the *$N$-adic valuation* of α is m, or $\log_N(\alpha) = m$. If $\alpha = 0$, then $\log_N(\alpha) = \infty$.

Definition 3.2.2. If $\alpha \in \mathbb{Z}_N$, then the *N-adic norm* of α is $\|\alpha\|_N = N^{-m}$ where $m = \log_N(\alpha)$.

This paper will be careful not to confuse $\log_N$ in $\mathbb{Z}_N$ with $\log_N$ in $\mathbb{R}$.

Example 3.2.3. Let the $\alpha = 00001011011\cdots \in \mathbb{Z}_N$. Then $\log_2(\alpha) = 4$.

3.3 Constructing Sequences Determining $\frac{b}{a}$ in $\mathbb{Z}_N$

For any rational number b/a, a relatively prime to N, there exists a sequence $(x_n) \to b/a \in \mathbb{Z}_N$. At this point, it is worth using the digit representation for integers in $\mathbb{Z}_N$. So $(x_n) = \{x_0,\ x_0 + x_1 N,\ \ldots,\ x_0 + \cdots + x_i N^i,\ \ldots\}$ and $b/a = x_0 x_1 \ldots x_i \ldots$. Rather than finding $a^{-1} \pmod{N^{i+1}}$ to determine each x_i, it is not too difficult for every i to find $\sum_{k=0}^{i} x_k N^k$ such that

$$b \equiv a \sum_{k=0}^{i} x_k N^k \pmod{N^{i+1}}. \tag{16}$$

Then,

$$x_i = \frac{\sum_{k=0}^{i} x_k N^k - \sum_{k=0}^{i-1} x_k N^k}{N^i}. \tag{17}$$

Nearly all of the digits for any rational number in $\mathbb{Z}_N$ can also be found using powers of N^{-1}, which is much simpler to analyze than the brute force search for the digits mentioned above.

Theorem 3.3.1. *Let $u_0, q, N \in \mathbb{Z}$, where q is relatively prime to N, $|u_0| < q$, and $q = -q_0 + \sum_{i=1}^{r} q_i N^i$ for $0 \le q_i < N$. Define $\alpha = u_0/q \in \mathbb{Z}_N$ such that $\alpha = \sum_{i=0}^{\infty} a_i N^i$ for $0 \le a_i < N$. Also, define $u_k \in \mathbb{Z}$ such that $u_k/q = \sum_{i=k}^{\infty} a_i N^{i-k} \in \mathbb{Z}_N$ and $\gamma \equiv N^{-1} \pmod{q}$. Then, there exist u_k for every $k \ge 0$ such that*

$$a_k \equiv q^{-1} u_k \pmod{N}. \tag{18}$$

If $-q < u_0 < 0$, then $u_k \in \{-q, \ldots, -1\}$ for $k \ge 0$. Otherwise, for $k > \lfloor \log_N(q) \rfloor = r$, $u_k \in \{-q, \ldots, -1\}$.

Let $\omega \in \{-q, \ldots, -1\}$ such that $\omega \equiv \gamma^k u_0 \pmod{q}$. Then for $k > \lfloor \log_N(q) \rfloor = r$, or if $-q < u_0 < 0$, then $k \ge 0$,

$$a_k \equiv q^{-1} \omega \pmod{N}. \tag{19}$$

Proof. Write u_0/q in terms of u_k.

$$\begin{aligned} \frac{u_0}{q} &= a_0 + N\frac{u_1}{q} = a_0 + a_1 N + N^2 \frac{u_2}{q} = \ldots \\ &= \sum_{i=0}^{k-1} a_i N^i + N^k \frac{u_k}{q} \quad \forall k \ge 1. \end{aligned} \tag{20}$$

Rewrite (20) to be

$$N^k u_k = u_0 - q\left(\sum_{i=0}^{k-1} a_i N^i\right) \quad \forall k \ge 1 \tag{21}$$

Then $|u_0| < q$ and $0 \leq a_i < p$ from the assumptions and equation (21). These imply for all $k \geq 1$, $|u_0| = |q\sum_{i=0}^{k-1} a_i N^i + N^k u_k| < q$. Then,

$$-q\left(\frac{1+\sum_{i=0}^{k-1} a_i N^i}{N^k}\right) < \ u_k < q\left(\frac{1-\sum_{i=0}^{k-1} a_i N^i}{N^k}\right).$$

u_k may only be greater than zero when $\frac{1-\sum_{i=0}^{k-1} a_i N^i}{N^k}$ is greater than zero. This only occurs when the sequence $(a_0, \ldots, a_j) = (0, \ldots, 0)$ for $j \geq 0$. Such a sequence occurs if and only if $u_0 \geq 0$ and $u_0 \equiv 0 \pmod{N^i}$ for $0 \leq i \leq j$, $j \geq 0$. This is clear from the construction of N-adic sequences for rational numbers. Therefore u_k may only be greater than zero if $u_0 \geq 0$ and $u_0 \equiv 0 \pmod{N^i}$ for $0 \leq i \leq j$, $j \geq 0$. The lower bound for u_k is greater than $-q$. This is clear because $\frac{1+\sum_{i=0}^{k-1} a_i N^i}{N^k} \leq 1$. Therefore,

$$-q < u_k < 0 \ \text{ for } \ -q < u_0 < 0.$$

If $0 \leq u_0 < q$, then the upper bound remains unchanged.

$$-q < u_k < q\left(\frac{1-\sum_{i=0}^{k-1} a_i N^i}{N^k}\right) \text{ for } 0 \leq u_0 < q$$

There is still work to be done on the upper bound.

$$\begin{aligned}
&0 \leq \sum_{i=0}^{k-1} a_i N^i < N^k \ \textit{for } k \geq 1 \\
\Rightarrow &-q\left(\sum_{i=0}^{k-1} a_i N^i\right) \leq 0 \\
\Rightarrow &\ u_0 - q\left(\sum_{i=0}^{k-1} a_i N^i\right) < q \\
\Rightarrow &\ N^k u_k < q \\
\Rightarrow &\ u_k < \frac{q}{N^k}.
\end{aligned}$$

For $k > \lfloor \log_N(q) \rfloor = r$, $|q/N^k| < 1$. Therefore, $-q < u_k < 0$ for $0 \leq u_0 < q$ and $k > r$. Further lowering the upperbound, if $u_k = 0$, then $u_0/q = \sum_{i=0}^{k-1} a_i N^i + 0$. This implies u_0/q is a rational integer, which is not true. Noting finally that u_k must be an integer. If $|u_0| < q$ and $u_0 < 0$, or $|u_0| < q$, $u_0 \geq 0$, and $k > \lfloor \log_N(q) \rfloor = r$, then

$$u_k \in \{-q, \ldots, -1\}.$$

It has now been shown, for certain restrictions, u_k belongs to a specific set of representatives for the residue classes of $\mathbb{Z}/(q)$. Define $\gamma \equiv N^{-1} \pmod{q}$. Reducing Equation (21) modulo q shows that

$$u_k \equiv \gamma u_{k-1} \pmod{q}. \tag{22}$$

Since this is true for all k greater than or equal to 1, it is clear that

$$u_k \equiv \gamma^k u_0 \pmod{q}. \tag{23}$$

Reducing (21) modulo p shows that

$$a_k \equiv q^{-1} u_k \pmod{N}. \tag{24}$$

Define $\omega \equiv \gamma^k u_0 \pmod{q}$, and $\rho \equiv q^{-1} \pmod{N}$. Finally, if $|u_0| < q$ and $u_0 < 0$, or $|u_0| < q$, $u_0 \geq 0$, and $k > \lfloor \log_N(q) \rfloor = r$, then

$$a_k \equiv \rho\omega \pmod{N}. \tag{25}$$

□

Corollary 3.3.2. *Let $0 \leq u_0 < q$. Define j to be the greatest integer such that $u_0 \equiv 0 \pmod{p^j}$. Then the following are true:*

i. $(a_0, \ldots, a_{j-1}) = (0, \ldots, 0)$

ii. $u_k > 0$ *for* $k = j$

iii. $u_k \not\equiv 0 \pmod{p}$

Theorem 3.3.1 shows that for $-q < u_0 < 0$, there is a sequence of numerators $\{u_k\}$ directly related to the sequence of digits $\{a_k\}$ for $u_0/q \in \mathbb{Z}_N$. This is provides a more powerful tool for the analysis of sequences generated by Feedback with Carry Shift Registers which are discussed in the next section.

4 Shift Registers

The *feedback with carry shift register* (or FCSR) is a type of shift register used in stream ciphers. Though there exist attacks against this type of shift register by itself, it is possible to combine FCSRs together in ways that no known attacks are useful. This however does not guarantee security because the combining of the FCSRs greatly increases the complexity of the cryptanalysis on the stream cipher. Despite the security challenges, the speed of an FCSR implementation is very attractive for engineers of hardware based cryptosystems.

This section will be used as an introduction to finite state machines and feedback with carry shift registers. The FCSR will be considered in the binary case and analyzed using $\mathbb{Z}_2$. This analysis is then extended to the case when FCSRs produce bent sequences, the theme of Section 5.

4.1 Finite State Machines

It is appropriate to preface the discussion about FCSRs with general finite state machines. Solomon W. Golomb's book *Shift Register Sequences* [6], written in 1967 and revised in 1982, established a definition of finite state machines and shift registers used in much of the literature today.

Definition 4.1.1. A *finite state machine* consists of a finite collection of *states* K, which sequentially accepts a sequence of *inputs* from a finite set A, and produces a sequence of *outputs* from a finite set B. Moreover, there is an *output function* μ which computes the present output as a fixed function of present input and present state, and a *next state function* δ which computes the next states as a fixed function of present input and present state. In a more mathematical manner, μ and δ are defined such that

$$\mu : K \times A \to B \qquad \mu(k_n, a_n) = b_n \tag{26}$$

$$\delta : K \times A \to K \qquad \delta(k_n, a_n) = k_{n+1} \tag{27}$$

The most fundamental observation by Golomb is the following theorem.

Theorem 4.1.2. *If the input sequence to a finite state machine is eventually periodic, then the output sequence is eventually periodic.*

Proof. Let p be the period of the inputs once the machine becomes periodic at time t. Then, for $h > 0$ and $c > t$, $a_c = a_{c+hp}$. Since K is finite, there must be $r > s > t$ such that, for some $h > 0$ such that,

$$k_{r+1} = \delta(k_r, a_r) = \delta(k_s, a_{r+hp}) = k_{s+1}.$$

It should also be clear that $a_{r+i} = a_{r+i+hp}$ for $h > 0$. So by induction, $\forall i > 0$

$$k_{r+i+1} = \delta(k_{r+i}, a_{r+i}) = \delta(k_{s+i}, a_{r+i+hp}) = k_{s+i+1}$$

Finally, this proves $b_{r+i+1} = b_{s+i+1}$. Thus, the eventual period of this machine is $r - s$. □

The next object defined is called an N-ary n-stage machine. It can be used to represent any finite state machine. It is also a natural generalization of shift registers, so thinking of finite state machines in the context of N-ary n-state machines will make the transition to talking about shift registers much smoother.

Definition 4.1.3. Choose $n, m, r \in \mathbb{N}$. Then define a finite state machine with the following sets:

1. $D = \{0, \ldots, N-1\}$. This set contains the *N-ary digits* of the machine.
2. $K = \{\sum_{i=0}^{n} x_i N^i : x_i \in D\}$. This set contains the *N-ary states* of the machine.
3. $A = \{\sum_{i=0}^{m} y_i N^i : y_i \in D\}$. This set contains the *N-ary inputs* of the machine.
4. $B = \{\sum_{i=0}^{r} z_i N^i : z_i \in D\}$. This set contains the *N-ary outputs* of the machine.
5. $F = \{f_i(x_0, \ldots, x_n, y_0, \ldots, y_m) : 0 \leq i < n\}$. This set contains the *N-ary next state functions* of the machine.

6. $G = \{g_i(x_0, \ldots, x_n, y_0, \ldots, y_m) : 0 \leq i < r\}$. This set contains the *N-ary output functions* of the machine.

The next state and output are determined from the current state and input by the following equations:

$$x_i^* = f_i(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}) \quad 0 \leq i < n \tag{28}$$

$$z_i = g_i(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}) \quad 0 \leq i < r \tag{29}$$

This finite state machine is called an *N-ary n-stage machine* and will be denoted by $\mathcal{M}(N, n, m, r)$.

By making the state the input to the machine as well, this machine becomes autonomous in the sense that it no longer needs outside input. Then each new state and output is based on the previous state of the machine. For $N = 2$, f_i and g_i are Boolean functions on $n+m$ variables. A binary n-stage machine can be defined by $n + r$ Boolean functions each on $n + m$ variables.

4.2 Feedback with Carry Shift Registers

In the set of autonomous finite state machine is a type of machine called a *shift register*. The variables making up the state of a shift register pass their values directly to the next variable in the state until the value is pushed out of the register as the output.

Definition 4.2.1. Let $\mathcal{M}(2, n, -1, 0)$ be an binary n-stage machine with no input and exactly 1 output value. Also, let $g, f_i \in \mathcal{BF}_n$ where the states $(x_0, \cdots, x_{n-1})$ are the domain of g and f_i, $f_i(x) = x_{i+1}$ for $0 \leq i \leq n-2$, and $g(x) = x_0$. This type of machine will be called an *n-stage binary shift register* denoted by $\mathcal{SR}(2, n)$.

When the function $f_{n-1} \in \mathcal{BF}_n$ is linear, then $\mathcal{SR}(2, n)$ is called a *linear feedback shift register*. An LFSR is drawn in Figure 2. This is the case where $f_{n-1} = \sum_{i=1}^{n} q_i x_{n-i}$ where each $x_i, q_i \in \mathbb{F}_2$. The q_i's are called *taps*. In computer science terms, to move forward in the sequence of states, each bit in the state of the register shifts to the right one spot and then the newest bit enters on the left end of the register and is the value given when each bit from the previous state is AND'ed with its corresponding tap and then XOR'ed with all the other AND'ed bit and taps.

Linear feedback shift registers are well-studied in [6]. By using the Berlekamp-Massey algorithm it is possible to recover the state of a given LFSR based on the output sequence. In fact given an LFSR output sequence with period $2^n - 1$, the Berlekamp-Massey algorithm will find a unique minimal-length LFSR which generates this output after the first $2n$ digits have been processed [14]. This algorithm is studied in more detail in [2].

In Figure 3, there is a memory cell attached to the linear feedback shift register which adds some complexity to the register. In the modified shift register shown in Figure 3, in each cycle, the whole number quotient of $\sum_{i=1}^{n} q_i x_{n-i}$

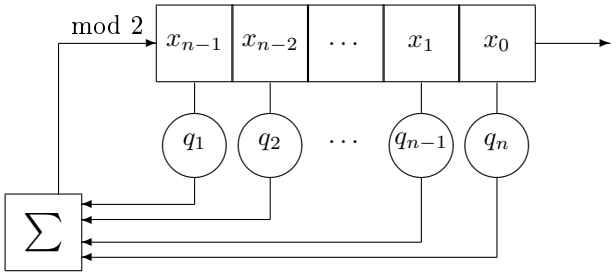

Figure 2: Linear Feedback Shift Register

is kept in the memory cell z. The memory cell from the previous state of the modified shift register is used to determined the sum modulo 2 for the newest bit in the state of the register. A shift register modified in this way is known as a *binary feedback with carry shift register*, or FCSR. For this paper, FCSRs will only be considered in the binary case. Many of the theorems do generalize the N-ary case, though sometimes it is necessar that N be prime.

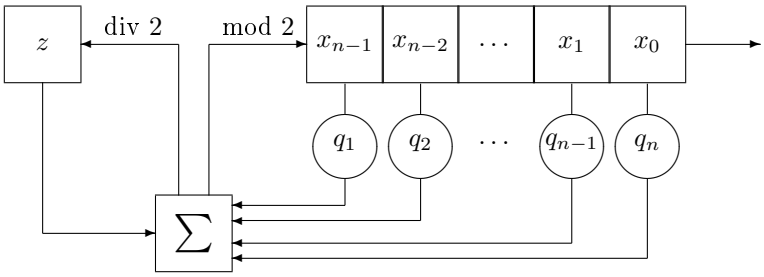

Figure 3: Binary Feedback with Carry Shift Register

Definition 4.2.2. Let $q_0, q_1, \ldots, q_n \in \mathbb{Z}/2$. A *binary feedback with carry shift register* of length n with *taps* $q_1, \ldots, q_n$ is a modified shift register whose states are collections

$$(x_0, x_1, \ldots, x_{n-1}; z) \text{ where } x_i \in \mathbb{F}_2 \text{ and } z \in \mathbb{Z}$$

where z is called the *memory cell*. The state changes according to the following rules:

1. Compute
$$\sigma = \sum_{i=1}^{n} q_i a_{n-i} + z.$$

2. The output is x_0.

3. Then the new state $(x_0, x_1, \ldots, x_{n-1}; z) = (x_1, \ldots, x_{n-1}, \sigma \pmod 2); \sigma(\text{div } 2)$.

Lemma 4.2.3. *If the sequence $\alpha = (a_0, a_1, \cdots)$ where $a_i \in \{0, 1\}$ and $a_i \equiv x_i \pmod 2$, and $\zeta = (z_{n-1}, z_n, z_{n+1}, \cdots)$ where each z_i is the value of the memory cell for the corresponding x_i, then these two sequences are related by the following linear recurrence*

$$a_k + 2z_k = q_1 a_{k-1} + \cdots + q_n a_{k-n} + z_{k-1} \text{ for } k \geq n. \tag{30}$$

5 Boolean Sequences

The interest of this paper is stream ciphers, and there are a few different ways to use bent functions in the implementation of a stream cipher. Sequences generated using bent functions have nice cryptographic properties because of their perfect nonlinearity. These sequences can be generated multiple ways. Two easy examples are a filtering function on a shift register producing an m-sequence or a shift register which uses n different shift registers as input into a bent function. These two techniques are discussed by Carlet [3]. Both of these constructions use input vectors from $\mathbb{F}_2^n$ in a psuedorandom order to generate the sequence. Before scrambling the input in this way, the sequences generated by lexicographic ordering of input vectors is considered.

Definition 5.0.4. Let (a_n) be a sequence. If T is the smallest integer such that $a_i = a_{i+T}$, then the *minimal period* of (a_n) is T.

Definition 5.0.5. Let $f \in \mathcal{BF}_n$ and $v_i \in \mathbb{F}_2^n$ such that $v_i = B^{-1}(i)$ for $0 \leq i < 2^n$. Then,

$$seq(f) = (f(v_0), f(v_1), \cdots, f(v_{2^n-1}), f(v_0), \cdots) \tag{31}$$

is a *lexicographical Boolean sequence.*

Defined in this way, all lexicographical Boolean sequences have a minimal period at most 2^n. Using the lexicographic ordering, the Boolean sequence generated will be repeated columns of the outputs for the Boolean function read from the truth table of the Boolean fucntion. For example, the lexicographic Boolean sequence of f in Table 2 is

$$(0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, \cdots).$$

Theorem 5.0.6. *The lexicographical Boolean sequence of a Bent function has a period exactly 2^n.*

Proof. For $x_i, \lambda_i \in \mathbb{F}_2$ and $0 \leq i \leq n-2$, define $(x, 1) = (x_0, \cdots, x_{n-2}, 1)$, $(x, 0) = (x_0, \cdots, x_{n-2}, 0)$, and $(\lambda, 1) = (\lambda_0, \cdots, \lambda_{n-2}, 1)$. Suppose $f \in \mathcal{BF}_n$

and $seq(f)$ has a period $T = 2^j < 2^n$. Then, $f(x,0) = f(x,1)$.

$$\begin{aligned}
c(\lambda,1) &= \frac{1}{2^{n/2}}\left(\sum_{x\in\mathbb{F}_2^{n-1}} (-1)^{f(x,0)+(x,0)\cdot(\lambda,1)} + (-1)^{f(x,1)+(x,1)\cdot(\lambda,1)}\right)\\
&= \frac{1}{2^{n/2}}\left(\sum_{x\in\mathbb{F}_2^{n-1}} (-1)^{f(x,0)}\left((-1)^{(x,0)\cdot(\lambda,1)} + (-1)^{(x,1)\cdot(\lambda,1)}\right)\right)\\
&= \frac{1}{2^{n/2}}\left(\sum_{x\in\mathbb{F}_2^{n-1}} (-1)^{f(x,0)}\left((-1)^{0\cdot 1} + (-1)^{1\cdot 1}\right)\right)\\
&= \frac{1}{2^{n/2}}\left(\sum_{x\in\mathbb{F}_2^{n-1}} (-1)^{f(x,0)}\cdot 0\right)\\
&= 0.
\end{aligned}$$

One of the Fourier coefficients of f must equal zero. Thus, f cannot be a Bent function. Clearly, every lexicographical Boolean sequence has a minimal period at most 2^n. Therefore, if g is a Bent function, then $seq(g)$ has period exactly 2^n. □

Boolean sequences will be considered as 2-adic exansions of rational numbers.

Definition 5.0.7. Let $f \in \mathcal{BF}_n$ and $v_i \in \mathbb{F}_2^n$ such that $v_i = B^{-1}(i)$ for $0 \leq i < 2^n$. Then,

$$\alpha_f = (f(v_0), f(v_0) + f(v_1)\cdot 2, \cdots, f(v_0) + \cdots + f(v_i)\cdot 2^i, \cdots) \tag{32}$$

where $\alpha_f \in \mathbb{Z}_2$ is called the *2-adic expansion* of f.

Lemma 5.0.8. *The digit representation of α_f is $seq(f)$.*

Recall the Maiorana-McFarland class of Boolean functions from Section 2.5, and consider the subset of these functions where $g(y) = 0$. Then the following theorem is true.

Theorem 5.0.9. $\log_2(\alpha_{x\cdot\pi(y)}) = 2^{n/2} + 2^{\bar{\pi}(y_0)}$

References

[1] Z. I. Borevich and I. R. Shafarevich, *Number theory*, Academic Press, 1966.

[2] T. B. Brock, *Linear feedback shift registers and cyclic codes in sage*, Honors paper (U.S.N.A. Department of Mathematics) (2006).

[3] C. Carlet, *Boolean functions for cryptography and error correcting codes*, Boolean Methods and Models (Y. Crama and P. L. Hammer, eds.), Cambridge University Press, 2006.

[4] T. W. Cusik and P. Stănică, *Cryptographic boolean functions and applications*, Elsevier, 2009.

[5] J. Đ. Golić, *Recent advances in stream cipher cryptanalysis*, Publications De L'Institut Mathématique **64** (1998), 183–204.

[6] S. W. Golomb, *Shift register sequences*, Aegean Park Press, 1982.

[7] M. Goresky and A. Klapper, *Algebraic shift register sequences*, Cambridge University Press, 2012.

[8] A. Klapper and M. Goresky, *Cryptoanalysis based on 2-adic rational approximation*, CRYPTO, 1995, pp. 262–273.

[9] ———, *Feedback shift registers, 2-adic span, and combiners with memory*, Journal of Cryptology **10** (1997), 111–147.

[10] ———, *Arithmetic correlations and walsh transforms*, IEEE Transactions on Information Theory **58** (2012), no. 1, 479–492.

[11] A. Klapper and J. Xu, *Algebraic feedback shift registers*, Theor. Comput. Sci. **226** (1999), no. 1-2, 61–92.

[12] ———, *Register synthesis for algebraic feedback shift registers based on non-primes*, Des. Codes Cryptography **31** (2004), no. 3, 227–250.

[13] N. Koblitz, *p-adic numbers, p-adic analysis, and zeta-functions*, Springer-Verlag, 1977.

[14] J. L. Massey, *Shift-register synthesis and bch decoding*, IEEE Transactions on Information Theory **15** (1969), 122–127.

[15] J. R. Munkres, *Topology: A first course*, Prentice Hall, 1975.

[16] T. Neumann, *Bent functions*, Ph.D. thesis, University of Kaiserslautern, May 2006.

[17] N. Nisan and M. Szegedy, *On the degree of boolean functions as real polynomials*, Computational Complexity **4** (1994), 301–313.

[18] A. A. Salnikov O. A. Logachev and V. V. Yashchenko, *Boolean functions in doing theory and cryptography*, American Mathematical Society, 2011.

[19] R. A. Reuppel, *Analysis and design of stream ciphers*, Springer-Verlag, 1986.

[20] O. S. Rothaus, *On "bent" functions*, Journal of Combinatorial Theory **20** (1976), no. 3, 300–305.

[21] W. J. Townsend and M. A. Thornton, *Walsh spectrum computations using cayley graphs*, IEEE Midwest Symposium on Circuits and Systems, August 2001, pp. 110–113.

[22] W. Trappe and L. C. Washington, *Introduction to cryptography with coding theory*, 2nd ed., Pearson Education, 2006.