Feedback with Carry Shift Registers and Bent Sequences

MIDN 1/C Charles Celerier

May 4, 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 f-filtered bent sequences with large 2-adic valuation. Rearrangements of these sequences could be effective for filtering the states of feedback with carry shift registers (FCSRs) in stream ciphers. The non-linearity of f-filtered bent sequences could provide resistance for FCSR-based stream ciphers in register synthesis attacks. In this paper, we show that it is possible to compute the 2-adic valuation of a bent sequence generated by a Maiorana-McFarland class Boolean function.

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 Figure 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

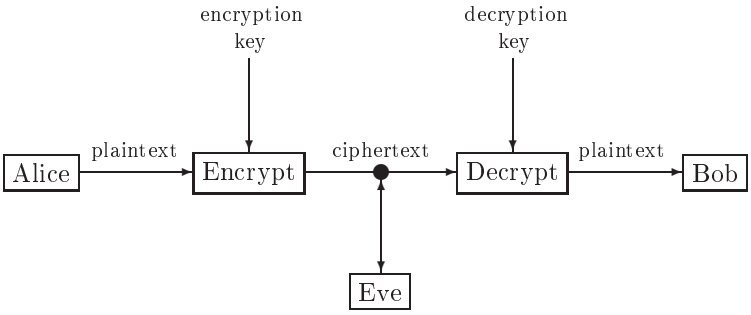

Figure 1: The basic communication scenario for cryptography

Alice and Bob. Many of these systems have been broken because of the amount 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. 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.

This paper will explore bent sequences for their potential strength in *stream ciphers*. According to Rueppel in [19], a *stream cipher* 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. 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 secretly 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 difficult to create perfectly random sequences, it is feasible to get close. A sequence which is close to being random is called a *pseudorandom sequence*. In [6], Golomb lists three randomness postulates on periodic binary sequences:

R-1. In every period, the number of 0s is nearly equal to the number of 1s.

R-2. In every period, half the runs have length one, one-fourth have length two, one-eighth have length three, etc., as long as the number of runs so

indicated exceeds 1. Moreover, for each of these lengths, there are equally many runs of 0s and of 1s.

R-3. The auto-correlation function $C(\tau)$ is two-valued. Explicitly

$$C(\tau) = \frac{1}{p}\sum_{n=1}^{p}(-1)^{a_n a_{n+\tau}} = \begin{cases} 1 \text{if } \tau = 0, \\ K \text{if } 0 < \tau < p. \end{cases}$$

These postulates today have become a measure of how close a sequence is to being considered pseudorandom. The claim is that binary sequences with extremely long periods would be nearly indistinguishable from perfectly random binary sequences.

In the case of this paper, the *f-filtered Boolean sequence* defined in Section 5 is considered in the context of pseudorandom sequences and FCSRs. The main result is the calculated 2-adic valuation of f-filtered bent sequences generated by the Maiorana-McFarland class of Boolean functions.

In Section 2, Boolean functions are first introduced, and the section concludes with the definition of the Walsh transform and bent functions. In Section 3, the N-adic ring is defined and a few properties of 2-adic sequences are discussed. The 2-adic valuation defined in this section is critical to understanding the main result in Section 5. Section 4 defines the FCSR following a fundamental discussion on finite state machines and n-state shift registers. Two shift register synthesis algorithms are given to demonstrate the necessity of adding a non-linear component to an FCSR before using it as a secure stream cipher. Finally, Section 5 defines what this paper calls *f-filtered Boolean sequences* and proves the main result on the 2-adic valuation of bent sequences generated by the functions in the Maiorana-McFarland class.

Most of the examples in this paper are shown using `Sage v4.8`. Methods for the examples are from the class `sage.crypto.boolean_functions` which comes with the current version of `Sage`. In addition, the script `afsr.sage` is attached for using many of the custom functions written for this paper. This script is available at `https://github.com/celerier/oslo/blob/master/sage/afsr.sage`.

Acknowledgments: I thank Professor David Joyner for his advice and expertise which allowed me to write this paper. Without his help, this work would not have been possible. I would also like to thank Professor Mark Goresky of the Princeton Institute for Advanced Study for his kind encouragement and for suggesting the problem addressed by the main result of this thesis. I also thank the Math Department for travel funds to attend the National Conference for Undergraduate Research at Weber State University in Ogden, Utah.

2 Boolean Functions

This section will establish the definition of a Boolean function and how to write these functions as polynomials. The goal is to introduce the Walsh transform,

and important tools used in cryptography. The Walsh transform measures the non-linearity of a Boolean function. This section concludes with the definition of bent functions which in terms of the Walsh transform are "perfectly non-linear" Boolean functions. 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 logical *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 its own inverse. In fact, $(\mathbb{F}_2, \oplus, \cdot)$ 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 vectors in $\mathbb{F}_2^n$ will be known as *bits*. For two vectors $x, y \in \mathbb{F}_2^n$ where $x = (x_0, \ldots, x_{n-1})$ and $y = (y_0, \ldots, y_{n-1})$, 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.

Sage

```
sage: V=GF(2)^3
sage: a=V([1,0,1]); b=V([0,1,1]);
sage: a+b
(1, 1, 0)
sage: a.dot_product(b)
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:

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

If we define the vectors $v_i \in \mathbb{F}_2^n$ by $v_i = B^{-1}(i)$ 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 *binary order*. This ordering is the standard ordering used by Sage to list vectors in $\mathbb{F}_2^n$.

Example 2.1.2.

Sage

```
sage: V=GF(2)^3
sage: V.list()
[(0, 0, 0), (1, 0, 0), (0, 1, 0), (1, 1, 0), (0, 0, 1), (1, 0, 1),
     (0, 1, 1), (1, 1, 1)]
sage: [GFn_to_integer(v) for v in V]
[0, 1, 2, 3, 4, 5, 6, 7]
```

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 2.1.3. Let $x, y \in \mathbb{F}_2^n$. Then $wt : \mathbb{F}_2^n \to \{0, \ldots, n-1\}$ is defined by

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

and $d : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \{0, \ldots, n-1\}$ is defined by

$$d(x, y) := \mathrm{wt}(x + y).$$

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

Definition 2.1.4. Let $x \in \mathbb{F}_2^n$. Then $\mathrm{supp} : \mathbb{F}_2^n \to 2^{\{0,\ldots,n-1\}}$ is defined by

$$\mathrm{supp}(x) := \{i \in \{0, \ldots, n-1\} : x_i = 1\}$$

Example 2.1.5.

Sage

```
sage: V=GF(2)^5
sage: a=V([0,1,1,0,1]); b=V([0,0,1,1,0])
sage: Hamming_weight(a)
3
sage: Hamming_weight(b)
2
sage: a.support()
[1, 2, 4]
sage: b.support()
[2, 3]
sage: Hamming_weight(a+b)
3
```

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 vector space.

Proposition 2.1.6. *Let $x \in \mathbb{F}_2^n$. Then*

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

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

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

□

Example 2.1.7.

Sage

```
sage: V=GF(2)^6
sage: [sum([(-1)^(x.dot_product(y)) for y in V]) for x in V]
[64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0]
```

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

Definition 2.1.8. 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 $\mathcal{BF}_n$.

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

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 $\mathbb{F}_2^n$:

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

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

[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.

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 $\mathrm{wt}(f)$ atomic Boolean functions.

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 $\mathbb{F}_2[x_0, \ldots, x_{n-1}]/(x_0^2 \oplus x_0, \ldots, x_{n-1}^2 \oplus x_{n-1})$. To properly represent an atomic Boolean function, a polynomial must equal 1 at only one vector $(x_0, \ldots, x_{n-1})$. Recall the support function from Definition 2.1.4. Then the polynomial respresenting each Boolean function is as follows:

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

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

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

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 $\mathcal{BF}_4$

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

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

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

□

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,

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

This result is easily verified in Sage:

Sage

```
sage: f=BooleanFunction([0,1,1,0,1,0,1,0,0,0,1,0,0,0,1,1])
sage: f.algebraic_normal_form()
x0*x1*x2*x3 + x0*x1*x3 + x0*x3 + x0 + x1*x2*x3 + x1*x2 + x1 + x2*
    x3 + x2
```

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 $\mathbb{F}_2[x_0,\ldots,x_{n-1}]/(x_0^2\oplus x_0,\ldots,x_{n-1}^2\oplus x_{n-1})$. Let $f\in\mathcal{BF}_n$. Then there exists a unique set of $a_I\in\mathbb{F}_2$, $I\in 2^{\{0,\ldots,n-1\}}$, such that*

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

Proof.

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

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 measures the non-linearity of a Boolean function by determining the distance between a given Boolean function $f(x)$ and a linear function $\lambda\cdot x$. The Walsh transform is similar to the discrete Fourier transform and is in fact used to obtain the Fourier coefficients for a Boolean function. 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]. 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 for all 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

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

It is clear from the proof of Lemma 2.3.2 that all the characters of $\mathbb{F}_2^n$ correspond to all of the linear functions in $\mathcal{BF}_n$.

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

Lemma 2.3.4. *Every character χ_λ belongs to $\hat{\mathcal{BF}_n}$.*

Proof. This is trivially true. □

Lemma 2.3.5. *If*

$$W = \text{span}_{\mathbb{R}}(\hat{\mathcal{BF}_n}) = \left\{ \sum_{i \in I} a_i \hat{f}_i : \hat{f}_i \in \hat{\mathcal{BF}_n},\ a_i \in \mathbb{R},\ \text{and } |I| < \infty \right\},$$

then the $\hat{\mathbb{F}_2^n}$ forms an orthonormal basis of W.

Proof. The atomic Boolean functions correspond to a basis of W. Thus, the dimension of $\hat{\mathcal{BF}_n}$ is at most 2^n. The 2^n characters of $\mathbb{F}_2^n$ are elements of $\hat{\mathcal{BF}_n}$, by Lemma 2.3.4.

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

Therefore, the characters of $\mathbb{F}_2^n$ are orthonormal in W. Since the dimension of W is at most 2^n, and there are 2^n characters, $\hat{\mathbb{F}_2^n}$ forms an orthonormal basis of W. □

Definition 2.3.6. Let $f \in \mathcal{BF}_n$ and $\lambda \in \mathbb{F}_2^n$. Then the *Walsh transform* of f is defined by:

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

Let

$$\mathcal{W}(f) = (\mathcal{W}_f(\lambda_0), \ldots, \mathcal{W}_f(\lambda_{2^n-1})),$$

where $\lambda_i = B^{-1}(i)$, be the *Walsh spectrum* of f.

Example 2.3.7. Here are some examples of the Walsh spectrums of different Boolean functions:

Sage

```
sage: f0=BooleanFunction([0,1,1,0,1,0,1,0,0,0,1,0,0,0,1,1])
sage: f0.walsh_hadamard_transform()
(-2, 6, -6, -6, -2, -2, 2, -6, 2, 2, 6, -2, 2, -6, -2, -2)
sage: f0.algebraic_normal_form( )
x0*x1*x2*x3 + x0*x1*x3 + x0*x3 + x0 + x1*x2*x3 + x1*x2 + x1 + x2*
    x3 + x2
sage:
sage: f1=BooleanFunction([1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0])
sage: f1.walsh_hadamard_transform()
(0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
sage: f1.algebraic_normal_form()
x0 + 1
sage:
sage: f2=BooleanFunction([1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1])
sage: f2.walsh_hadamard_transform()
(0, 0, 0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
sage: f2.algebraic_normal_form()
x0 + x1 + 1
sage:
sage: f3=BooleanFunction([1,0,0,1,1,0,0,1,1,0,0,1,1,0,1,1])
sage: f3.walsh_hadamard_transform()
(2, 2, -2, 14, -2, -2, 2, 2, -2, -2, 2, 2, 2, 2, -2, -2)
sage: f3.algebraic_normal_form( )
x0*x1*x2*x3 + x0 + x1*x2*x3 + x1 + 1
```

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 2.3.8. *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 $c(0)$:

$$\begin{aligned} c(0) &= \frac{1}{2^{n/2}} \sum_{x\in\mathbb{F}_2^n} (-1)^{f(x)} \\ &= \frac{1}{2^{n/2}} \big((2^n - \mathrm{wt}(f)) - \mathrm{wt}(f)\big) \\ \\ \Rightarrow \mathrm{wt}(f) &= 2^{n-1} - 2^{n/2-1}c(0). \end{aligned} \tag{9}$$

Example 2.3.9. There are 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.

For large $|\mathcal{W}_f(\lambda)|$, the Hamming distance between f and an affine function in $\mathcal{BF}_n$ is small.

2.4 Bent Functions

These functions are useful in cryptographic applications because they add resistance to differential attacks as a result of being "perfectly non-linear". As mentioned before, these were defined by Rothaus in 1976. The main result in Section 5 is given on bent functions because of their important cryptographic property of being as far away as possible from every affine function in $\mathcal{BF}_n$.

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

Proposition 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,

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

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 $1 = -1 + 2$,

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

Thus, $\mathrm{wt}(g(x))$ 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. □

The set of all bent functions is not known. There however are a few basic constructions which are known. The Rothaus construction and Maiorana-McFarland class of Boolean functions are presented here as two of the known constructions for bent functions.

2.4.1 Rothaus Construction of Bent Functions

In one of the first papers written about bent functions, Rothaus identified two large general classes of bent function on $\mathcal{BF}_n$, $n = 2k$. The simpler of the two is presented here:

Proposition 2.4.3. *Let n be even, $x_1, y_1, \ldots, x_k, y_k$ be independent variables, and $P(x) \in \mathcal{BF}_{n/2}$ (so $P(x)$ is a function $n/2$ variables). Then the polynomial $Q(x,y) \in \mathcal{BF}_n$ given by*

$$Q(x,y) = x_1 y_1 + x_2 y_2 + \cdots + x_k y_k + P(x) \tag{10}$$

is bent.

The proof that this function is bent can be found in Rothaus' paper [20].

Example 2.4.4. Both f and g in this example are bent according to Rothaus.

Sage

```
sage: B=BooleanPolynomialRing(6,'x')
sage: B.inject_variables(verbose=false)
sage: f=BooleanFunction(x0*x1+x2*x3+x4*x5)
sage: f.truth_table(format='int')
(0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0,
     0, 1, 0,
0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
    0, 1, 1,
1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1)
sage: f.is_bent()
True
sage: g=f+BooleanFunction(x0*x2+x0)
sage: g.truth_table(format='int')
(0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
     0, 1, 0, 1,
0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
    1, 0, 1, 1, 1,
1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1)
sage: g.is_bent()
True
```

2.4.2 Maiorana-McFarland Class Construction of Bent Functions

The Maiorana-McFarland class (or M-M class) of Boolean functions is a generalized Rothaus construction of bent functions. It uses a permutation function π on half of the input variables to each function.

Definition 2.4.5. Let $\pi : \mathbb{F}_2^k \to \mathbb{F}_2^k$ be the linear transformation on $\mathbb{F}_2^n$ represented by the matrix $\pi \in M_{k\times k}(\mathbb{F}_2)$ such that the matrix π has exactly k non-zero entries arranged so that every row and column has exactly one non-zero entry. Then π is a *permutation matrix*.

Example 2.4.6.

Sage

```
sage: pi0=matrix(GF(2),[[0,0,1],[1,0,0],[0,1,0]])
sage: pi0
[0 0 1]
[1 0 0]
[0 1 0]
sage: V=GF(2)^3
sage: x=V([0,1,1]); y=V([1,0,0])
sage: (matrix(x)).transpose()
[0]
[1]
[1]
sage: pi0*(matrix(x)).transpose()
[1]
[0]
[1]
sage: (matrix(y)).transpose(
[1]
[0]
```

```
[0]
sage: pi0*(matrix(y)).transpose( )
[0]
[1]
[0]
```

For every permutation function π on $\mathbb{F}_2^k$ the *component permutation function* $\bar{\pi}$ can defined such that for $0 \leq i \leq k-1$,

$$\bar{\pi}(i) = j \text{ where } \pi(e_i) = e_j \text{ for basis vectors } e_i, e_j \in \mathbb{F}_2^k.$$

Example 2.4.7. If $\pi_0 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ and $\pi_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, then $\bar{\pi}_0$ is the permutation cycle $(0\,1\,2)$ and $\bar{\pi}_1$ is the permutation cycle $(0\,1)(2)$.

A simple bent function construction is accomplished by using permutations like the ones defined here. These Boolean functions belong to the *Maiorana-McFarland original class*. This is the set $\mathcal{M}$ which contains all Boolean functions on $\mathbb{F}_2^n = \{(x, y) = (x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}) : x, y \in \mathbb{F}_2^{n/2}\}$, 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 $\mathbb{F}_2^{n/2}$.

Proposition 2.4.8. [3]
All functions in the Maiorana-McFarland class of Boolean functions are bent.

The proof of these functions being bent can be found in [3].

Example 2.4.9. Both f and g in this example are in the Maiorana-McFarland class of Boolean functions.

Sage

```
sage: f=BooleanFunction(x0*x4+x1*x3+x2*x5)
sage: f.truth_table(format='int')
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1,
     0, 1, 0,
1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
    0, 0, 1,
0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1)
sage: f.is_bent()
True
sage: g=f+BooleanFunction(x0*x2+x0)
sage: g.is_bent()
True
```

3 N-adic Integers

This section introduces the N-adic integer ring and defines the *2-adic valuation* which is critical to the main result of the paper. The 2-adic integers are very closely connected with the binary sequences generated by FCSRs which are discussed in Section 4.

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

$$x_{i+1} \equiv x_i \pmod{N^{i+1}} \quad \forall i \geq 0. \tag{11}$$

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

$$x_i \equiv x'_i \pmod{N^{i+1}} \quad \forall i \geq 0. \tag{12}$$

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:

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

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

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

The sequences for (y_n) and (z_n) satisfy equation (11) 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 (12).

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

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. $(a_n) = (a_0,\ a_0 + a_1 \cdot N,\ \ldots,\ a_0 + \cdots + a_i \cdot N^i,\ \ldots) : 0 \leq a_i < N \quad \forall i \geq 0$

will be called *canonical*.

The sequence $(a_0, a_1, a_2, \ldots, a_i, \ldots)$ is the *digit representation* of α. When $N < 10$, the digits are usually written adjacent to one another. Another equivalent representation of α is the *power series representation* where $\alpha = \sum_{i=0}^{\infty} a_i N^i$ where the a_i's are from the digit representation of α.

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:

$\beta = 3164\cdots = 3 + 1 \cdot 7 + 6 \cdot 7^2 + 4 \cdot 7^3 + \cdots$, then the canonical sequence $(b_n) \to \beta$ is

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

$\gamma = 0164\cdots = 0 + 1 \cdot 7 + 6 \cdot 7^2 + 4 \cdot 7^3 + \cdots$, then the canonical sequence $(c_n) \to \gamma$ is

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

$\delta = 5031\cdots = 5 + 0 \cdot 7 + 3 \cdot 7^2 + 1 \cdot 7^3 + \cdots$, 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 $(x_n) \to \alpha \in \mathbb{Z}_N$.

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

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

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

$(x_n) = (x_n) \cdot (1, 1, \dots) = (1, 1, \dots) \cdot (x_n)$ 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:

$$\begin{aligned}a(b + c) &= ab + ac \\ (b + c)a &= ba + ca\end{aligned}$$

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*

$$\begin{aligned}(x_n)(y_n) &= (x_0y_0, \dots, x_iy_i, \dots)\\ &= (y_0x_0, \dots, y_ix_i, \dots)\\ &= (y_n)(x_n).\end{aligned}$$

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

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

$$\begin{aligned}(x_n)((y_n) + (z_n)) &= (x_n)(y_0 + z_0, \dots, y_i + z_i, \dots)\\ &= (x_0(y_0 + z_0), \dots, x_i(y_i + z_i), \dots)\\ &= (x_0y_0 + x_0z_0, \dots, x_iy_i + x_iz_i, \dots)\\ &= (x_n)(y_n) + (x_n)(z_n).\end{aligned}$$

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.

□

So that the power series representations of N-adic numbers make sense in the discussion of FCSRs in Section 4, equivalence is shown between the defined N-adic addition and multiplication and the usual addition and multiplication of power series representations of N-adic integers.

Lemma 3.1.9. *Addition and multiplication of canonical sequences of α and β are equivalent to the usual addition and multiplication of the power series representations for α and β.*

Proof. Let $\mathbf{a} = (a_0, a_0 + a_1N, \dots, a_0 + a_1N + \cdots + a_kN^k, \dots)$ and $\mathbf{b} = (b_0, b_0 + b_1N, \dots, b_0 + b_1N + \cdots + b_kN^k, \dots)$. And α and β be represented by power series so that

$$\alpha = \sum_{i=0}^{\infty} a_iN^i \text{ and } \beta = \sum_{i=0}^{\infty} b_iN^i.$$

Rewrite $\mathbf{a} = (\sum_{i=0}^{i} a_i, \dots, \sum_{i=0}^{k} a_i 2^i, \dots)$ and $\mathbf{b} = (\sum_{i=0}^{i} b_i, \dots, \sum_{i=0}^{k} b_i 2^i, \dots)$. By the defined N-adic addition,

$$\begin{aligned}
\mathbf{a} + \mathbf{b} &= (\sum_{i=0}^{0} a_i + b_i, \dots, \sum_{i=0}^{k} (a_i + b_i) 2^i, \dots) \\
&\to \sum_{i=0}^{\infty} (a_i + b_i) 2^i \\
&= \sum_{i=0}^{\infty} a_i N^i + \sum_{i=0}^{\infty} b_i N^i \\
&= \alpha + \beta.
\end{aligned}$$

Clearly addition is equivalent. By the defined N-adic multiplication,

$$\begin{aligned}
\mathbf{a} \cdot \mathbf{b} &= (\sum_{i=0}^{0} a_i \cdot b_i, \dots, \sum_{i=0}^{k} \sum_{j=0}^{k} (a_i b_j) 2^{i+j}, \dots) \\
&= (a_0 \cdot b_0, \dots, \sum_{i+j \le k} (a_i b_j) 2^{i+j}, \dots) \\
&\to \sum_{k=0}^{\infty} \left(\sum_{i+j=k} a_i b_j 2^k \right) \\
&= \left(\sum_{i=0}^{\infty} a_i 2^i \right) \left(\sum_{j=0}^{\infty} b_j 2^j \right) \\
&= \alpha' \cdot \beta'.
\end{aligned}$$

Therfore, multiplication is equivalent as well. □

Theorem 3.1.10. *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

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

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 (11)

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

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 $b \in \mathbb{Z}_N$, $b/a = a^{-1}b \in \mathbb{Z}_N$.

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

$$b \equiv a \sum_{k=0}^{i} x_k N^k \pmod{N^{i+1}}. \tag{14}$$

Then,

$$x_i = \frac{\sum_{k=0}^{i} x_k N^k - \sum_{k=0}^{i-1} x_k N^k}{N^i}. \tag{15}$$

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. The following theorem is a slightly modified version of Theorem 10 in a draft of [7] written by Andrew Klapper and Mark Goresky.

Theorem 3.1.11. *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 $0 \leq q_i < N$. Define $\alpha = u_0/q \in \mathbb{Z}_N$ such that $\alpha = \sum_{i=0}^{\infty} a_i N^i$ for $0 \leq a_i < N$. 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 $k \geq 0$ such that*

$$a_k \equiv q^{-1} u_k \pmod{N}. \tag{16}$$

If $-q < u_0 < 0$, then $u_k \in \{-q, \ldots, -1\}$ for $k \geq 0$. 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 $k \geq 0$,

$$a_k \equiv q^{-1} \omega \pmod{N}. \tag{17}$$

Proof. Write u_0/q in terms of u_k.

$$\begin{aligned}\frac{u_0}{q} &= a_0 + N\frac{u_1}{q} = a_0 + a_1 N + N^2\frac{u_2}{q} = \dots \\ &= \sum_{i=0}^{k-1} a_i N^i + N^k \frac{u_k}{q} \quad \forall k \geq 1. \end{aligned} \tag{18}$$

Rewrite (18) to be

$$N^k u_k = u_0 - q\left(\sum_{i=0}^{k-1} a_i N^i\right) \quad \forall k \geq 1 \tag{19}$$

Then $|u_0| < q$ and $0 \leq a_i < p$ from the assumptions and equation (19). 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 $(a_0, \dots, a_j) = (0, \dots, 0)$ 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.

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

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 (19) modulo q shows that

$$u_k \equiv \gamma u_{k-1} \pmod{q}. \tag{20}$$

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

$$u_k \equiv \gamma^k u_0 \pmod{q}. \tag{21}$$

Reducing (19) modulo p shows that

$$a_k \equiv q^{-1} u_k \pmod{N}. \tag{22}$$

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

$$a_k \equiv \rho\omega \pmod{N}. \tag{23}$$

□

Corollary 3.1.12. *Let $0 \leq u_0 < q$. Define j to be the greatest integer such that $u_0 \equiv 0 \pmod{N^j}$. Then the following are true:*

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

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

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

Theorem 3.1.11 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 $u_0/q \in \mathbb{Z}_N$. The sequence of numerators for a given FCSR with connection integer q reveals all of the possible strictly periodic states of the register, each identified by u_k/q. This provides an interesting tool for the analysis of different FCSRs.

3.2 2-adic Integers

Now the transition is made to discussing the 2-adic integers whose digit sequences are infinite sequences of 0s and 1s. This brings the discussion closer the previous sections which dealt with vectors and functions defined on $\mathbb{F}_2^n$. The 2-adic integers will act as a bridge between the results on Boolean functions and the FCSRs which are defined in the next section.

Proposition 3.2.1. [9] *There is a one-to-one correspondence between rational numbers $\alpha = p/q \in \mathbb{Z}_2$ (where q is odd) and eventually periodic binary sequences $\mathbf{a} = (a_0, a_1, \dots)$, which associates to each such rational number α its digit representation $\mathbf{a}$. The sequence $\mathbf{a}$ is strictly periodic if and only if $\alpha \leq 0$ and $|\alpha| < 1$.*

Proof. Consider the strictly periodic case first. Let $\mathbf{a} = (a_0, a_1, \dots)$ be a strictly periodic sequence of period T. Set $\alpha = \mathbf{a}$. Computing in $\mathbb{Z}_2$,

$$2^T \alpha = \sum_{i=0}^{\infty} a_i 2^{i+T} = \sum_{i=0}^{\infty} a_{i+T} 2^{i+T} = \sum_{i=T}^{\infty} a_i 2^i = \alpha - \sum_{i=0}^{T-1} a_i 2^i.$$

Hence

$$\alpha = -\frac{\sum_{i=0}^{T-1} a_i 2^i}{2^T - 1} \tag{24}$$

is a negative rational number. Write $\alpha = p/q$ as a fraction reduced to lowest terms with q positive. Then q is odd, $p \leq 0$, and $|p| < q$.

On the other hand, suppose that $\alpha = p/q$ is given in lowest terms with q an odd positive integer, $p \leq 0$, and $|p| < q$. Let T be the smallest integer such that $2^T \equiv 1 \pmod{q}$. Such a T exists because q is odd. Then $2^T - 1$ is divisble by q, so set $s = (2^T - 1)/q$. Because $p = q \sum_{i=0}^{\infty} a_i 2^i$

$$s \cdot (-p) = s(-q \sum_{i=0}^{\infty} a_i 2^i) = (1 - 2^T)(\sum_{i=0}^{\infty} a_i 2^i) = \sum_{i=0}^{\infty} a_i 2^i - \sum_{i=T}^{\infty} a_i 2^i = \sum_{i=0}^{T-1} a_i 2^i$$

Thus $\alpha = s \cdot p/(2^T - 1) = -(\sum_{i=0}^{T-1} a_i 2^i)/(2^T - 1)$. It directly follows that $\alpha = 2^T \alpha + \sum_{i=0}^{T-1} a_i 2^i$, implying that the digits sequence $\mathbf{a}$ of α is strictly periodic.

Now suppose that $\alpha = p/q$ is an arbitrary rational number. Let $M = \lceil \alpha \rceil$ be the next largest integer. If $M \geq 0$, then its digit sequence ends in an infinite string of 0s. If $M < 0$, then its digit sequence ends in an infinite string of 1s. However, $\alpha = M + p'/q$ where $p' \leq 0$ and $|p'| < q$, so the digit sequence of p'/q is strictly periodic. It follows that the digit sequence $\mathbf{a}$ of $\alpha = M + p'/q$ must be eventually periodic.

On the other hand, an eventually periodic sequence $\mathbf{a} = (a_0, a_1, \dots)$ corresponds to a rational number $\alpha = \mathbf{a}$ because it is given by a finite transient term $\sum_{i=0}^{k-1} a_i 2^i$ (for some nonnegative integer k) plus a periodic term, $\sum_{i=k}^{\infty} a_i 2^i = 2^k \sum_{i=0}^{\infty} a_{j+k} 2^j$, both of which are rational numbers. □

The 2-adic valuation is needed for the main result. It is defined here.

Definition 3.2.2. Let $\alpha = (a_n) \in \mathbb{Z}_2 \backslash (0)$. If m is the smallest number in $\mathbb{N}$ such that $a_m \not\equiv 0 \pmod{2^{m+1}}$, then the *2-adic valuation* of α is m, or $\log_2(\alpha) = m$. If $\alpha = 0$, then $\log_2(\alpha) = \infty$.

Definition 3.2.3. If $\alpha \in \mathbb{Z}_2$, then the *2-adic norm* of α is $\|\alpha\|_2 = 2^{-m}$ where $m = \log_2(\alpha)$.

This paper will be careful not to confuse $\log_2$ in $\mathbb{Z}_N$ with $\log_2$ in $\mathbb{R}$.

Example 3.2.4. Let $\alpha = 00001011011\cdots \in \mathbb{Z}_2$. Then $\log_2(\alpha) = 4$.

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 bent sequences which could be generated by FCSRs, which leads to the main result in 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

$$\mu : K \times A \to B \qquad \mu(k_n, a_n) = b_n \tag{25}$$

$$\delta : K \times A \to K \qquad \delta(k_n, a_n) = k_{n+1} \tag{26}$$

The most fundamental observation by Golomb is the following proposition. Its result guarantees the periodicity of any finite state machine with eventually periodic input.

Proposition 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. $K = \{\sum_{i=0}^{n-1} x_i N^i : x_i \in D\}$. This set contains the *N-ary states* of the machine.
3. $A = \{\sum_{i=0}^{m-1} y_i N^i : y_i \in D\}$. This set contains the *N-ary inputs* of the machine.
4. $B = \{\sum_{i=0}^{r-1} z_i N^i : z_i \in D\}$. This set contains the *N-ary outputs* of the machine.
5. $\Delta = \{\delta_i(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}) : 0 \leq i < n\}$ where $\delta_i : K \times A \to D$. This set contains the *N-ary next state functions* of the machine.
6. $M = \{\mu_i(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}) : 0 \leq i < r\}$. $\mu_i : K \times A \to D$. 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:

$$x_i^* = \delta_i(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}) \quad 0 \leq i < n \tag{27}$$

$$z_i = \mu_i(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}) \quad 0 \leq i < r \tag{28}$$

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 machines 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. Here, what was referred as an n-stage machine will now be called an r-stage machine as n will be used to index the output sequences of FCSRs.

Definition 4.2.1. Let $\mathcal{M}(2, r, -1, 0)$ be an binary r-stage machine with no input and exactly 1 output value. Also, let $g, f_i \in \mathcal{BF}_n$ where the states $(x_0, \ldots, x_{r-1})$ 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 denoted by $\mathcal{SR}(2, r)$.

When the function $f_{r-1} \in \mathcal{BF}_r$ is linear, then $\mathcal{SR}(2, r)$ is called a *linear feedback shift register*. An LFSR is drawn in Figure 2. This is the case where $f_{r-1} = \sum_{i=1}^{r} q_i x_{r-i}$ 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.

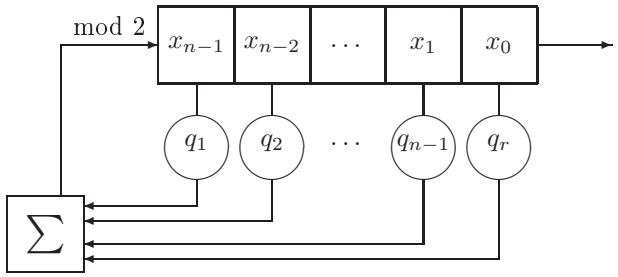

Figure 2: Linear Feedback Shift Register

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 $2^r - 1$, 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 $\sum_{i=1}^{r} q_i x_{r-i}$ 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 necessary that N be prime.

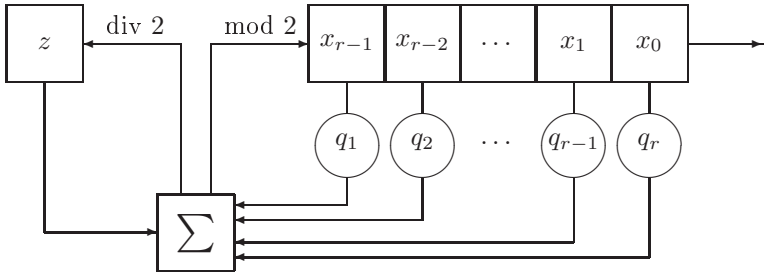

Figure 3: Binary Feedback with Carry Shift Register

Definition 4.2.2. Let $q_1, \ldots, q_r \in 0, 1 \subset \mathbb{Z}$ and $q_0 = -1$. A *binary feedback with carry shift register* of length n with *taps* $q_1, \ldots, q_r$ is a modified shift register whose states are collections

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

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

1. Compute
$$\sigma_n = \sum_{i=1}^{r} q_i a_{n-i} + z_{n-1}.$$
2. The output is x_0.
3. Then the new state $(x_0, x_1, \ldots, x_{n-1}; z) = (x_1, \ldots, x_{n-1}, \sigma_n \pmod 2; \sigma_n(\text{div } 2))$.

Lemma 4.2.3. *If the sequence $\alpha = (a_0, a_1, \ldots)$ where $a_i \in \{0, 1\}$ and $a_i \equiv x_i \pmod 2$, and $\zeta = (z_{n-1}, z_n, z_{n+1}, \ldots)$ 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*

$$a_k + 2z_k = q_1 a_{k-1} + \cdots + q_r a_{k-r} + z_{k-1} \text{ for } k \geq r. \tag{29}$$

Recall from Proposition that an FCSR must be eventually periodic. By Proposition , the sequence generated by every FCSR can then be represented by a rational number in $\mathbb{Z}_2$. This rational number, or sequence generated, is entirely determined by the initial state and the taps of the register.

Proposition 4.2.4. [9] *Let $q_1, \ldots, q_r$ be the taps, z_{r-1} be the initial memory, and $a_{r-1}, \ldots, a_1, a_0$ be the initial state of an FCSR. Define $q = 1 + \sum_{i=1}^{r} q_i 2^i$ and*

$$p = \left(\sum_{i=0}^{r-1} \sum_{j=0}^{i} q_j a_{i-j} 2^i \right) - z_{r-1} 2^r.$$

Then the output sequence $\mathbf{a}$ *of this FCSR is the digit representation of the 2-adic integer*

$$\alpha = \sum_{i=0}^{\infty} a_i 2^i = p/q. \tag{30}$$

Proof. Consider the transition from one state of the FCSR to the next. Suppose that, for some given state, the value of the memory is z_{n-1} and that the contents of the register is given by the r bits $a_{n-1}, \ldots, a_{n-r}$. The next state of the register is determined by calculating

$$\sigma_n = z_{n-1} + \sum_{i=1}^{r} q_i a_{n-i},$$

writing the new memory contents as $z_n = \sigma_n (\text{div } 2)$, and writing the new contents of the state $a_n = \sigma_n \pmod 2$. As stated in Lemma 4.2.3, these equations may be combined into the expression

$$\sigma_n = 2z_n + a_n.$$

It follows that

$$a_n = \sum_{i=1}^{r} q_i a_{n-i} + (z_{n-1} - 2z_n), \tag{31}$$

for $n \geq r$. Now, by substituting Equation (31) into Equation (30),

$$\begin{aligned} \alpha &= a_0 + a_1 2 + \cdots + a_{r-1} 2^{r-1} + \sum_{n=r}^{\infty} a_n 2^n \\ &= x + \sum_{n=r}^{\infty} \left(\sum_{i=1}^{r} q_i a_{n-i} \right) 2^n + \sum_{n=r}^{\infty} (z_{n-1} - 2z_n) 2^n. \end{aligned} \tag{32}$$

where $x = \sum_{i=0}^{r-1} a_i 2^i$ is the integer represented by the initial state of the register. The second summation in Equation (32) cancels except for the first term, z_{r-1},

leaving

$$\begin{aligned}
\alpha &= x + z_{r-1}2^r + \sum_{n=r}^{\infty}\sum_{i=1}^{r} q_i 2^i a_{n-i} 2^{n-i} \\
&= x + z_{r-1}2^r + \sum_{i=1}^{r} q_i 2^i \left(\sum_{n=r}^{\infty} a_{n-i} 2^{n-i}\right) \\
&= x + z_{r-1}2^r + \sum_{i=1}^{r} q_i 2^i (\alpha - (a_0 2^0 + a_1 2^1 + \cdots + a_{r-i-1}2^{r-i-1})) \\
&= x + z_{r-1}2^r + \alpha \sum_{i=1}^{r} q_i 2^i - \sum_{i=1}^{r-1}\sum_{j=0}^{r-i-1} q_i 2^i a_j 2^j.
\end{aligned}$$

(where the inner sum is empty, hence zero, when $i = r$ in the third line). These equations give

$$\begin{aligned}
\alpha &= \frac{x + z_{r-1}2^r - \sum_{i=1}^{r-1}\sum_{j=0}^{r-i-1} q_i 2^i a_j 2^j}{1 - \sum_{i=1}^{r} q_i 2^i} \\
&= \frac{\sum_{i=0}^{r-1}\sum_{j=0}^{r-i-1} q_i 2^i a_j 2^j - z_{r-1}2^r}{q}
\end{aligned}$$

since $q_0 = -1$. The double summation is over all pairs of integers $0 \le i, j \le r-1$ with $i + j \le r - 1$. Setting $k = i + j$ gives

$$\alpha = \frac{\left(\sum_{k=0}^{r-1}\sum_{i=0}^{k} q_i a_{k-i} 2^k\right) - z_{r-1}2^r}{q} = \frac{p}{q} \tag{33}$$

as claimed. □

Corollary 4.2.5. *Changing the memory by b changes the value of α by $-b2^r/q$. If $\alpha = p/q < 0$, then the initial memory $z_{r-1} \ge 0$.*

Proof. The first statement follows trivially from Equation (33).

The second statemtent is not as obvious. If $q < 0$, then the numerator must be positive for p/q to be negative. Since

$$z_{r-1}2^r \ge \sum_{k=0}^{r-1}\sum_{i=0}^{k} q_i a_{k-i} 2^k \ge 0, \tag{34}$$

this implies $z_{r-1} = 0$. If $q > 0$, then the numerator must negative. By Equation (34), $z_{r-1} > 0$. Therefore, $z_{r-1} \ge 0$. □

It can also be shown that the memory cell of every FCSR is bounded and eventually lies between 0 and wt(q + 1), for $q > 0$.

Proposition 4.2.6. [9]

Let $w = \mathrm{wt}(q+1)$. If an FCSR is in a periodic state, then the memory is in the range $0 \le z < w$. If the initial memory $z_{n-1} \ge w$, then it will monotonically decrease and will arrive in the range $0 \le z < w$ within $\lfloor \log_2(z_{n-1} - w) \rfloor + r$ steps. If the initial memory $z_{n-1} < 0$, then it will monotonically increase and will arrive in the range $0 \le z < w$ within $\lceil \log_2(|z_{n-1}|) \rceil + r$ steps. (The logarithm functions in this proposition will be real-valued.)

Proof. First, observe that if the initial memory value z_{n-1} lies in the range $0 \le z_{n-1} < w$, then the same will be true for all later values of the memory. This follows from Definition 4.2.2 because $\sigma_n = \sum_{i=1}^{r} q_i a_{n-i} + z_{n-1} \le w + z_{n-1} < 2w$. So $z_n = \lfloor \sigma_n / 2 \rfloor < w$.

By the same argument, if the initial memory value is $z_{n-1} = w$, then the later values of memory will be no greater than w; but in this case, within r steps, the memory will drop below w (and will remain so thereafter) for the following reason. If the memory does not decrease (i.e. $z_n = w$), then this means that a 1 appeared at *all* the tapped cells, that $\sigma_n = 2w$, and that $x_n = \sigma_n \pmod 2 = 0$ was fed into the register. The value of σ will fall below $2w$ when this 0 reaches the first tapped cell (if not before), at which time we will have $z = \lfloor \sigma / 2 \rfloor < w$.

Moreover, if we initialize an FCSR with a larger memory value, $z_{n-1} > w$, then with each step, the excess $e_{n-1} = z_{n-1} - w$ will become reduced by a factor of $1/2$, that is $e_n \le \lfloor e_{n-1}/2 \rfloor$. So after $\lfloor \log_2(z_{n-1} - w) \rfloor + 1$ steps, the memory will be no more than w. This follows from Definition 4.2.2 which gives

$$e_n = z_n - w = \left\lfloor \frac{\sigma_n}{2} \right\rfloor - w \le \left\lfloor \frac{w + w + e_{n-1}}{2} \right\rfloor - w = \left\lfloor \frac{e_{n-1}}{2} \right\rfloor.$$

Now consider the case of negative initial memory, $z_{n-1} < 0$. By Definition 4.2.2, it is possible that $\sigma_n \ge 0$, in which case the next memory value will be $z_n \ge 0$ (where it will remain thereafter). So suppose that $\sigma_n < 0$. Then, again by Definition 4.2.2,

$$|z_n| \le \frac{|\sigma_n| + 1}{2} \le \frac{|z_{n-1}| + 1}{2}.$$

Iterating this formula, it is easy to see that after $K = \lceil \log_2(|z_{n-1}|) \rceil$ steps, either the memory z has become nonnegative, or else

$$|z| \le \frac{z_{n-1}}{2^K} + \frac{1}{2^K} + \frac{1}{2^{K-1}} + \cdots + \frac{1}{2} < 2$$

, in which case the memory must be $m = -1$. There is a single situation in which the memory can remain at -1 forever: if there are no feedback taps on the shift register (so $q = -1$). In this case, the memory will feed 1s into the shift register forever. However, we assumed that $q > 0$ to rule out this possibility. If $q > 0$, then as soon as a nonzero feedback occurs, the memory will become nonnegative, where it will remain thereafter. □

Prosition 4.2.6 shows that eventually every FCSR reaches a point where there are a finite number of inputs from the memory cells and of states. This means

that every FCSR eventually satisfies the definition of a finite state machine. As a result, the output of every FCSR is eventually periodic. In Section 3, it was shown than every eventually periodic sequence of 0s and 1s corresponds to an $\alpha = p/q \in \mathbb{Z}_2$. This fact makes FCSRs extremely vulnerable to rational approximation algorithms.

4.3 FCSR Synthesis

The problem of synthesis lies in the following question: Given an eventually periodic sequence of 0s and 1s generated by an FCSR, can you find a and b such that sequence generated is equivalent to digit representation of $\frac{a}{b} \in \mathbb{Z}_2$. If there are no constraints on a and b, then at least a period of the sequence must be known to solve the problem. However, every FCSR is limited to a certain number of $p/q \in \mathbb{Z}_2$ that it can generate become of memory restrictions. The rational approximation algorithm shown in this paper uses the fact that only so many possibilities exist for a given FCSR to show that it will eventually reach the correct approximation in a finite number of steps. As a quick example of this, consider the FCSR in Figure 4.3. By Proposition 4.2.4, it should be clear that the initial states, taps, and memory completely determine the 2-adic integer represented by the sequence generated. Both the initial states and taps are finite, and the memory is bounded at the initial stage of the register.

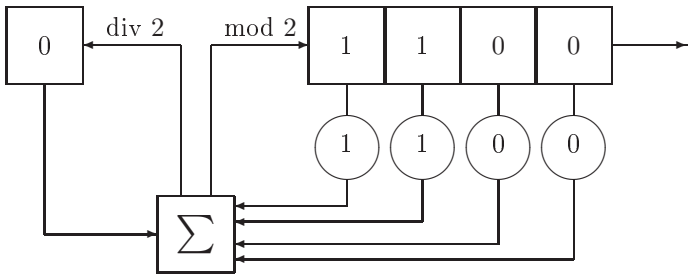

Figure 4: FCSR which generate the sequence $\frac{-4}{5} = 00110011001100110011\cdots$

The FCSR in Figure 4.3 is meant as a simple visualization how after fooling with all of the possible initial states and tap arrangements, the size of the memory becomes the only means to create new possibilities in the sequence generated.

4.4 Xu's Rational Approximation Algorithm

It turns out that given a sequence generated by an FCSR it is easy to reproduce to FCSR from which it came from. This is why using FCSRs by themselves do not generate secure stream ciphers. If used as a stream cipher, the key for the FCSR would be the taps and the initial state. All an attacker would need

is a relatively short piece of the output sequence to reveal every part of the key. This ciphertext only attack is a complete break of the stream cipher. In fact, any arrangement of taps and initial state of an FCSR can be revealed very quickly.

In Goresky and Klapper's book [7], they describe in pseudocode Xu's rational approximation algorithm for π-adic sequences in any ring R. A demonstration of Xu's algorithm is presented here in the context of the ring $\mathbb{Z}_2$. The algorithm takes as input the first n terms of an N-adic sequence $\mathbf{a} = (a_0, a_1, \dots)$ associated to a rational number $\alpha = p/q \in \mathbb{Z}_2$ and outputs a rational number $\alpha_n = p'/q'$ whose first n terms match $\mathbf{a}$. Running Xu's algorithm for small n can sometimes result in α_n which are far from α, but in the examples below, it is easy to see that as n grows large, Xu's algorithm eventually reaches the correct α. In fact, for large enough n, Xu's algorithm will reach the correct α.

Example 4.4.1. This example uses the function `rational_synthesis_xu` from the Sage script `afsr.sage`. The source code for this function can be download from `https://github.com/celerier/oslo/blob/master/sage/afsr.sage`.

Sage

```
sage: adic_seq(-4,5,2,20)
(-4, 5, [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
    1])
sage: a=adic_seq(-4,5,2,20)[2]
sage: for i in range(3,20):
...:     print i,rational_synthesis_xu(a[0:i],2)
...:
3 (0, 1)
4 (-4, 53)
5 (-4, 53)
6 (-4, 53)
7 (-4, 5)
8 (-4, 5)
sage:
sage: adic_seq(-17,77,2,40)
(-17, 77, [1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
    0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
     1])
sage: b=adic_seq(-17,77,2,40)[2]
sage: for i in range(3,20):
...:     print i,rational_synthesis_xu(b[0:i],2)
...:
3 (-1, 9)
4 (-5, 17)
5 (-5, 17)
6 (1, 3)
7 (1, 3)
8 (1, 3)
9 (-13, 89)
10 (-167, 1419)
11 (-67, 183)
12 (-89, 885)
13 (-89, 885)
14 (-89, 885)
```

```
15 (359, 2229)
16 (359, 2229)
17 (359, 2229)
18 (359, 2229)
19 (359, 2229)
20 (-17, 77)
21 (-17, 77)
22 (-17, 77)
23 (-17, 77)
```

The N-adic sequences for $\alpha = \frac{-4}{5}$ and $\beta = \frac{-17}{77}$ have small periods, so Xu's algorithm quickly converges to the correct α and β.

Example 4.4.2. In this next example, the period of the approximated rational number $\gamma = \frac{-98}{2^{1000}-1}$ equals 1000.

Sage

```
sage:adic_seq(-98,2^1000-1,2,20)
(-98, 107150860718626732094842504906000..., [0, 1, 0, 0, 0, 1, 1,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
sage: d=adic_seq(-98,2^1000-1,2,10000)[2]
sage: for i in range(3,2000):
...:     print i,rational_synthesis_xu(d[0:i],2)
...:
3 (-2, 9)
4 (2, 1)
5 (2, 1)
6 (2, 1)
7 (-22, 69)
8 (-22, 69)
9 (-302, 1209)
10 (-302, 1209)
11 (-302, 1209)
12 (-302, 1209)
13 (-302, 1209)
14 (-134, 333)
15 (-134, 333)
16 (-134, 333)
17 (-134, 333)
18 (-134, 333)
19 (-2818, 10671)
20 (-2818, 10671)
21 (-2818, 10671)
22 (-2818, 10671)
23 (-2818, 10671)
24 (-26954, 85323)
25 (-26954, 85323)
26 (-26954, 85323)
27 (-26954, 85323)
28 (-26954, 85323)
29 (98, 1)
30 (98, 1)
31 (98, 1)
32 (98, 1)
.
.
```

```
.
2501 (-32524788108326247..., 180020230874340668557...)
2502 (-98, 10715086071862673209484250490600 0...)
```

The algorithm does not converge nearly as fast as it did from approximating α, but it eventually reaches the correct γ_n at $n = 2502$. It took approximately 1.25s for my computer to arrive at the correct approximation.

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 pseudorandom order to generate the sequence. Before scrambling the input in this way, the sequences generated by binary ordering of input vectors is considered.

Definition 5.0.3. Let (a_n) be a sequence. If T is the smallest positive integer such that $a_i = a_{i+T}$, then the *minimal period* of (a_n) is T.

Definition 5.0.4. 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,

$$seq(f) = (f(v_0), f(v_1), \ldots, f(v_{2^n-1}), f(v_0), \ldots) \tag{35}$$

is a *f-filtered Boolean sequence*.

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

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

Theorem 5.0.5. *The f-filtered 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 $(x,1) = (x_0, \ldots, x_{n-2}, 1)$, $(x,0) = (x_0, \ldots, x_{n-2}, 0)$, and $(\lambda, 1) = (\lambda_0, \ldots, \lambda_{n-2}, 1)$. Suppose $f \in \mathcal{BF}_n$ and $seq(f)$ has a period $T = 2^j < 2^n$. Then, $f(x,0) = f(x,1)$.

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

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

$$= \frac{1}{2^{n/2}} \left(\sum_{x \in \mathbb{F}_2^{n-1}} (-1)^{f(x,0)} \left((-1)^{0\cdot 1} + (-1)^{1\cdot 1} \right) \right)$$

$$= \frac{1}{2^{n/2}} \left(\sum_{x \in \mathbb{F}_2^{n-1}} (-1)^{f(x,0)} \cdot 0 \right)$$

$$= 0.$$

One of the Fourier coefficients of f must equal zero. Thus, f cannot be a bent function. Clearly, every f-filtered 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 5.0.6. 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,

$$\alpha_f = (f(v_0), f(v_0) + f(v_1) \cdot 2, \ldots, f(v_0) + \cdots + f(v_i) \cdot 2^i, \ldots) \tag{36}$$

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

Lemma 5.0.7. *The digit representation of α_f is $seq(f)$.*

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

Theorem 5.0.8. $\log_2(\alpha_f) = 2^{n/2} + 2^{\bar{\pi}(0)}$ *where* $f = x \cdot \pi(y)$.

Proof. Let $f(x, y) = x \cdot \pi(y)$ and $(x, y) = (x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}) \in \mathbb{F}_2^n$ where $x, y \in \mathbb{F}_2^{n/2}$. Define $v_i = (x, y)_i = B^{-1}(i)$ for $0 \leq i \leq 2^n - 1$. Then $y = 0$ for $0 \leq i \leq 2^{n/2} - 1$ and $x = 0$ for $i = 2^{n/2}$. Thus, $f(v_i) = 0$ for $0 \leq i \leq 2^{n/2}$.

Now, $\log_2(\alpha_f) = \min\{i : f(v_i) = 1\}$. The claim is that $\min\{i : f(v_i) = 1\} = 2^{n/2} + 2^{\bar{\pi}(0)}$.

Let $A = \{(x, y)_i : 2^{n/2} \leq i \leq 2^{n/2+1} - 1\}$. Then A is the set of vectors in $\mathbb{F}_2^n$ where

$$y_k = \begin{cases} 1 \text{ if } k = 0 \\ 0 \text{ if } k > 0 \end{cases}$$

If $u = (x, y) \in A$, then

$$f(u) = x_{\bar{\pi}(0)} y_0 = \begin{cases} 1 \text{ if } x_{\bar{\pi}(0)} = 1 \\ 0 \text{ if } x_{\bar{\pi}(0)} = 0 \end{cases}$$

Then $f(u) = 1$ for exactly $2^{n/2-1}$ distinct elements $u \in A$.

Define (x', y') such that

$$x'_k = \begin{cases} 1 \text{ if } k = \bar{\pi}(0) \\ 0 \text{ if } k \neq \bar{\pi}(0) \end{cases} \qquad y'_k = \begin{cases} 1 \text{ if } k = 0 \\ 0 \text{ if } k \neq 0 \end{cases}$$

Then $B(x', y') \leq B(u)$ for all $u \in A$. Thus, $i = B(x', y')$ is the smallest i such that $f(v_i) = 1$ and $v_i \in A$. $f(v_i) = 0$ for $0 \leq i \leq 2^{n/2}$.

Therefore $\log_2(\alpha_f) = 2^{n/2} + 2^{\bar{\pi}(0)}$. □

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 coding 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.