Register Synthesis of Feedback with Carry Shift Registers

MIDN 1/C Charles Celerier

November 30, 2011

Abstract

Register synthesis algorithms can recover the state of a feedback with carry shift registers (FCSRs) using only a small portion of the generated sequence. This paper covers the background necessary to understand these algorithms and implements using Python scripts in SAGE. The background covered includes p-adic integers, finite state machines, and other properties of FCSRs.

1 Introduction

Finite state machines are useful for generating periodic sequences. These machines are composed of inputs, states, outputs, and a set of rules that determine the next state and output based on the current state and input. A shift register is a type of finite state machine that has a feedback function for its input based on the state of the machine. This feedback is what drives the autonomous behavior of the machine. The two general types of shift registers are linear feedback shift registers (LFSRs) and feedback with carry shift registers (FCSRs). The difference between the two types of registers is that the FCSR uses a memory cell for its input that remembers feedback from multiple previous states. Inputs for the LFSR are only based on a function of the last state. This paper will consider the generated sequences of FCSRs.

For an FCSR to be a useful stream cipher, it must produce sequences that are unpredictable if only a small portion of the sequence is known. To analyze the predictability of the output of an FCSR, this paper follows the work of Professor Andrew Klapper and Professor Mark Goresky by using N-adic integers, $\mathbb{Z}_N$, to represent the N-ary sequences generated by FCSRs. A property of the N-adic integers is that any eventually periodic sequence can be represented by a rational number $b/a \in \mathbb{Z}_N$ where a is relatively prime to N. Since FCSRs are finite state machines, their output is eventually periodic, which means the sequences generated by FCSRs can always be represented by rational numbers in $\mathbb{Z}_N$.

Register synthesis algorithms recover the state of an FCSR through the N-adic representation of the generated sequence. The goal of this paper is to

first lead the reader through the background of N-adic integers, finite state machines, and FCSRs, and then explain a register synthesis algorithm that will be implemented using a Python script in the open source algebraic computer system SAGE.

2 N-adic Integers

The notation used in the definition of the N-adic integers will follow the same notation used by Borevich and Shafarevich in Chapter 1 of *Number Theory*.

2.1 The N-adic Integer Ring

The commutativity of addition and multiplication is needed for the analysis of sequences generated by FCSRs. The proof for the N-adic integers being a commutative ring with an identity is given here.

Definition 2.1. Let N be an integer. Then the infinite integer sequence $\{x_n\}$ determines a *N-adic integer* α, or $\{x_n\} \to \alpha$, if

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

Two sequences $\{x_n\}$ and $\{x'_n\}$ determine the same N-adic integer if

$$x_i \equiv x'_i \pmod{N^{i+1}} \quad \forall i \geq 0. \tag{2}$$

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

Ordinary integers will be called *rational integers* and each rational integer x is associated with a N-adic integer, determined by the sequence $\{x,\ x,\ \ldots,\ x,\ \ldots\}$.

Example 1. Let $\{x_n\} \to \alpha \in \mathbb{Z}_3$. Then the first 5 terms of $\{x_n\}$ may look something like:

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

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 (1) 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 (2).

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

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 2.2. For a given N-adic integer α, a given sequence $\{a_n\}$ with the properties:

i. $\{a_n\} \to \alpha$

ii. $\{a_n\} = \{a_0,\ a_0 + a_1 \cdot N,\ \ldots,\ a_0 + \cdots + a_i \cdot N^i,\ \ldots\} : 0 \le a_i < N \quad \forall i \ge 0$

will be called *canonical*. The number $a_0a_1a_2 \ldots a_i \ldots$ is the *digit representation* of α.

Example 2. In Example 1, 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:

$\beta = 3164\ldots$, then the canonical sequence $\{b_n\} \to \beta$ is

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

$\gamma = 0164\ldots$, 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 2.3. 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 2.1. *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 paper defines a *ring* accroding to Fine, Gaglione, and Roesenberger and shows that $\mathbb{Z}_N$ is a *commutative ring with an identity.*

Definition 2.4. 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 2.1. *$\mathbb{Z}_N$ 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 2.1, $\alpha + \beta = \beta + \alpha$.

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 2.1, 0 is the additive identity.

4. *Existence of Additive Inverses*

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

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

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

5. *Associativity of Multiplication*

$$\begin{aligned}\{x_n\}(\{y_n\}\{z_n\}) &= \{x_n\}\{y_0z_0, \dots, y_iz_i, \dots\} \\ &= \{x_0(y_0z_0), \dots, x_i(y_iz_i), \dots\} \\ &= \{(x_0y_0)z_0, \dots, (x_iy_i)z_i, \dots\} \\ &= \{x_0y_0, \dots, x_iy_i, \dots\}\{z_n\} \\ &= (\{x_n\}\{y_n\})\{z_n\}.\end{aligned}$$

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 2.1, 1 is the multiplicative identity.

Properties 1 through 8 from Definition 2.4 are satisfied, so $\mathbb{Z}_N$ is a commutative ring with an identity.

□

2.2 Units

Here is a short of proof of what N-adic integer units look like.

Theorem 2.2. *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

$$x_i y_i \equiv 1 \pmod{N^{i+1}} \quad \forall i \geq 0. \tag{3}$$

In particular, $x_0 y_0 \equiv 1 \pmod{N}$ and hence $x_0 \not\equiv 0 \pmod{N}$. Thus, 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 (1)

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

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 $p \ \ \forall i \geq 0$, 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 $x_{i+1} \equiv x_i \pmod{p)^i}$ 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 this condition holds, then $a^{-1} \in \mathbb{Z}_N$. Then for any rational integer $b \in \mathbb{Z}_N$, $b/a = a^{-1}b \in \mathbb{Z}_N$. Here is a corollary about the digit representation of N-adic integers that immediately follows.

Corollary 2.1. *The N-adic integer α is a unit if and only if the first digit of α is non-zero.*

2.3 Constructing Sequences Determing $\frac{b}{a}$ 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

$$b \equiv a \sum_{k=0}^{i} x_k N^k \pmod{N^{i+1}}. \tag{4}$$

Then,

$$x_i = \frac{\sum_{k=0}^{i} x_k N^k - \sum_{k=0}^{i-1} x_k N^k}{N^k}. \tag{5}$$

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 2.3. *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*

$$a_k \equiv q^{-1} u_k \pmod{N}. \tag{6}$$

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 $k \geq 0$,

$$a_k \equiv q^{-1} \omega \pmod{N}. \tag{7}$$

Proof. Write u_0/q in terms of u_k.

$$\begin{aligned} \frac{u_0}{q} &= a_0 + N\frac{u_1}{q} = a_0 + a_1 N + N^2 \frac{u_2}{q} = \ldots \\ &= \sum_{i=0}^{k-1} a_i N^i + N^k \frac{u_k}{q} \quad \forall k \geq 1. \end{aligned} \tag{8}$$

Rewrite (8) to be

$$p^k u_k = u_0 - q\left(\sum_{i=0}^{k-1} a_i p^i \right) \quad \forall k \geq 1 \tag{9}$$

Then $|u_0| < q$ and $0 \leq a_i < p$ from the assumptions and equation (9). These imply for all $k \geq 1$, $|u_0| = |q \sum_{i=0}^{k-1} a_i p^i + p^k u_k| < q$. Then,

$$\begin{aligned} -q - q\sum_{i=0}^{k-1} a_i p^i < p^k u_k < q - q\sum_{i=0}^{k-1} a_i p^i \\ \Rightarrow -q\left(\frac{1 + \sum_{i=0}^{k-1} a_i p^i}{p^k} \right) < u_k < q\left(\frac{1 - \sum_{i=0}^{k-1} a_i p^i}{p^k} \right). \end{aligned}$$

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

$$-q < u_k < 0 \; for \; -q < u_0 < 0.$$

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

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

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

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

For $k > \lfloor \log_p(q) \rfloor = r$, $|q/p^k| < 1$. Therefore, $-q < u_k < 0$ for $0 \leq u_0 < q$ and $k > r$. Further lowering the upperbound, if $u_k = 0$, then $u_0/q = \sum_{i=0}^{k-1} a_i p^i + 0$. 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 $k > \lfloor \log_p(q) \rfloor = r$, 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 $\gamma \equiv p^{-1} \pmod{q}$. Reducing (9) modulo q shows that

$$u_k \equiv \gamma u_{k-1} \pmod{q}. \tag{10}$$

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

$$u_k \equiv \gamma^k u_0 \pmod{q}. \tag{11}$$

Reducing (9) modulo p shows that

$$a_k \equiv q^{-1} u_k \pmod{p}. \tag{12}$$

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

$$a_k \equiv \rho\omega \pmod{p}. \tag{13}$$

□

Corollary 2.2. *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. $j \leq \lfloor \log_p q \rfloor = r$

ii. $[a_0, \ldots, a_{j-1}] = [0, \ldots, 0]$

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

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

The results of this corollary are straightforward.

Theorem 2.3 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 $u_0/q \in \{z_n\}$. This is provides a more powerful tool for the analysis of the sequences generated by AFSRs. The results shown here fill in the gaps of the incorrect proof shown in Theorem 10 of Klapper and Xu's paper.

3 Finite State Machines

In 1967, Dr. Solomon W. Golomb published *Shift Register Sequences* establishing a definition of finite state machines and shift registers used in much of the literature today. This section will begin by defining the finite state machine according to Golomb and move toward the definition of generalized shift registers. In the next section, FCSRs will be defined with Golomb's style in mind.

Definition 3.1. A *finite state machine* consists of a finite collection of *states* K, 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

$$\mu : K \times A \to B \qquad \mu(k_n, a_n) = b_n \tag{14}$$

$$\delta : K \times A \to K \qquad \delta(k_n, a_n) = k_{n+1} \tag{15}$$

Golomb presents two important theorems about these machines. Both deal with the periodicity of any finite state machine, which include FCSRs. The theorems and proofs are presented here.

Theorem 3.1. *If the input to a finite state machine is eventually constant, then the output is eventually periodic.*

Proof. Let t be the time when the input becomes constant, so $a_t = a_{t+1} = \ldots$. Because K is a finite collection of states, there exists times $r > s > t$ such that $k_r = k_s$. Then, by induction, $\forall i > 0$,

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

Therefore,

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

Thus, the eventual period of this machine is $r - s$. □

Theorem 3.2. *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 $c > t$, $a_c = a_{c+hp}$. Similar to the proof of Theorem 3.1, using the fact that K is finite, there must be $r > s > t$ such that, for some $h > 0$,

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

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 3.2. Choose $n, m, r \in \mathbb{N}$. An *N-ary n-stage machine* consists of the following:

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:

$$x_i^* = f_i(x_0, \ldots, x_n, y_0, \ldots, y_m) \quad 0 \leq i < n \tag{16}$$

$$z_i = g_i(x_0, \ldots, x_n, y_0, \ldots, y_m) \quad 0 \leq i < r \tag{17}$$

4 SAGE

```
"""
Algebraic feedback shift register functions



AUTHOR:
   David Joyner, wdjoyner@gmail.com, 2011-10-19
   Charles Celerier, charles.celerier@gmail.com

LICENSE:
   Modified BSD

"""

def connection_element(Qs, p):
    """
    Input consists of
        Qs - r+1 elements from T ("taps")
        p - prime

    This function returns the connection element

    EXAMPLES:
        sage: p = 5; Qs = [2,2,1,2]
        sage: connection_element(Qs, p)
        283

    """
    r = len(Qs)-1
    c = -Qs[0]+sum([Qs[i]*p**i for i in range(1,r+1)])
    return c


def generating_quotient(As, Qs, m, p):
    """
    Input consists of
        S, T - complete sets of reps mod p
        As - r elements from S
        Qs - r+1 elements from T
        m - integer
        p - prime
        N - positive integer > r

    This function returns
```

```
    EXAMPLES:
        sage: p = 5; Qs = [2,2,1,2]; As = [1,2,2]; m = 4
        sage: generating_quotient(As, Qs, m, p)
        -487/283
        sage: alpha = Qp(5)(-487/283)
        sage: alpha.padded_list(20)
        [1, 2, 2, 1, 0, 1, 0, 1, 2, 0, 0, 0, 3, 0, 2, 0, 2, 4, 0, 0]

    This gives the same output as afsr does.

    """
    r = len(Qs)-1
    DS = sum([sum([Qs[i]*As[n-i]*p**n for i in range(1,n+1)]) for n in range(1,r)])
    u = -m*p**r - Qs[0]*sum([As[i]*p**i for i in range(r)]) + DS
    return u/connection_element(Qs, p)


def afsr(S, T, As, Qs, m, p, N):
    """
    Input consists of
        S, T - complete sets of reps mod p
        As - r elements from S
        Qs - r+1 elements from T
        m - integer
        p - prime
        N - positive integer > r

    This function returns a list of length N generated
    by the AFSR associated to (S, T, As, Qs, m, p), as in [KX]

    EXAMPLES:
        sage: p = 5; N = 20; S = range(p); T = S
        sage: As = [1,2,2]; Qs = [1,2,1,2]; m = 4
        sage: afsr(S, T, As, Qs, m, p, N)
        [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
        sage: As = [1,2,2]; Qs = [2,2,1,2]; m = 4
        sage: afsr(S, T, As, Qs, m, p, N)
        [1, 2, 2, 1, 0, 1, 0, 1, 2, 0, 0, 0, 3, 0, 2, 0, 2, 4, 0, 0]
        sage: Qp(5)(-487/283)
         1 + 2*5 + 2*5^2 + 5^3 + 5^5 + 5^7 + 2*5^8 + 3*5^12 + 2*5^14
  + 2*5^16 + 4*5^17 + O(5^20)
        sage: alpha.padded_list(20)
        [1, 2, 2, 1, 0, 1, 0, 1, 2, 0, 0, 0, 3, 0, 2, 0, 2, 4, 0, 0]
        sage: multiplicative_order(mod(5,283)) # the period
        282
```

```
REFERENCE:
    [KX] Klapper and Xu, "Algebraic feedback shift registers",
         Theoretical Com. Sci., vol 226, 1999,
         http://www.cs.uky.edu/~klapper/papers.html

"""
r = len(As)
# test if S and T will work
if len(S) <> p:
    raise NameError("arg S is not a complete set of reps mod ", p)
if len(T) <> p:
    raise NameError("arg T is not a complete set of reps mod ", p)
Zp = IntegerModRing(p)
S0 = [Zp(s) for s in S]
S0.sort()
if S0 <> [Zp(x) for x in range(p)]:
    raise NameError("arg S is not a complete set of reps mod ", p)
T0 = [Zp(t) for t in T]
T0.sort()
if T0 <> [Zp(x) for x in range(p)]:
    raise NameError("arg T is not a complete set of reps mod ", p)
# test if As are in S
for x in As:
    if not(x in S):
        raise NameError("arg As is not a set of fills for ", S)
# test if Qs are in T
for x in Qs:
    if not(x in T):
        raise NameError("arg Qs is not a set of taps for ", T)
if len(Qs) <> r+1:
    raise NameError("arg Qs is not a set of taps for ", T)
# force m to be an initial memory value
m = ZZ(m)
# force p to be a prime
p = next_prime(p-1)
# force N to be an integer
N = ZZ(N)
conn_elmt = -Qs[0]+sum([Qs[i]*p**i for i in range(1,r+1)])
newAs = As
new_m = m
a = S[0]
if N <= r:
    return As
for j in range(N-r):
```

```
        tau = sum([Qs[i]*newAs[-i] for i in range(1,r+1)]) + new_m
        for s in S:
            if Zp(tau) == Zp(s*Qs[0]):
                newAs.append(s)
                new_m = ZZ((tau-s*Qs[0])/p)
                #print j, tau, s, new_m, r
    return newAs


def afsr2(u, q, p, N):
    """
    Input consists of
        u, q - positive integers (q<p^(20))
        p - positive integer
        N - positive integer > r

    This function returns a triple:
     -  u,q in least denominator form,
     -  a list of length N generated by the AFSR associated
        to (u, q, p), as in Theorem 10, [KX])

    EXAMPLES:
        sage: adic_seq(u, q, p, N)
        (972, 1001, [2, 4, 3, 1, 1, 4, 0, 4, 0, 3, 1, 3, 4, 1, 3, 2, 4, 2, 3, 3])
        sage: afsr2(u, q, p, N)
        (972, 1001, [2, 4, 3, 1, 1, 4, 0, 4, 0, 3, 1, 3, 4, 1, 3, 2, 4, 2, 3, 3])

    REFERENCE:
        [KX] Klapper and Xu, "Algebraic feedback shift registers",
             Theoretical Com. Sci., vol 226, 1999,
             http://www.cs.uky.edu/~klapper/papers.html

    """
    S = range(p)
    U = range(q)
    G = gcd(u,q)
    u = ZZ(u/G)
    q = ZZ(q/G)
    gamma = 1
    for i in U:
        if i*p%q==1:
            gamma = i
    rho = 1
    for i in S:
        if i*q%p==1:
            rho = i
```

```
    aseq=[];useq=[];j=0;
    useq+=[(u*gamma**k)%q-q for k in range(N)]
    while (u%(p**(j+1))==0 and  u>=0):
        aseq+=[0]
        j+=1
    aseq+=[rho*((u*gamma**j)%q)%p]
    aseq+=[rho*useq[k]%p for k in range(j+1,N)]
    return u,q,aseq

#######################################

def index(x,M):
    """
    Inputs a non-zero integer and outputs the
    integer part of its log.

    """
    return int(log(abs(x))/log(M))

def height(x,y,M):
    """
    Inputs a pair of non-zero integers and outputs the
    max of the integer part of their log.

    """
    return int(max(log(abs(x))/log(M),log(abs(y))/log(M)))

def interpolation_set(M):
    """
    Returns the interpolation set associated to the
    modulus M.

    """
    if M==2:
        L = 5
    if M==3:
        L = 15
    if M>3:
        L = int(M**3/2)
    return range(-L,L+1)

def findst(h1,h2,i,M):
    """
    Solves h1*s+h2*t = 0 mod M^5.
```

```
    """
    P = interpolation_set(M)
    for s in P:
        for t in P:
            if (s*h1+t*h2)%M**(i+5) == 0:
                return (s,t)

def maxpower(u,q,M):
    """
    Returns max power of M dividing u,q

    EXAMPLES:
        sage: maxpower(1000, 200, 10)
        2
        sage: maxpower(1200,9,9)
        0


    """
    r = gcd(u,q)
    m = int(log(r,M))
    for t in range(2*m):
        if r%M**t != 0:
            return t-1
    return 0

def ifsinP(i,h,r,a,M,P):
    """"

    """
    B = 5
    ans = 0
    Pn0 = [x for x in P if x!=0]
    for s in Pn0:
        if s*(h[i]-a*r[i])%M**(i+B)==0:
            return s
    return 0

def adic_seq(u,q,x,N):
    """
    Returns N elements of the x-adic sequence for u/q,
    and u,q in least denominator form.

    Raises a ValueError if the denominator is divisible
    by the modulus x.
```

```
    EXAMPLES:
        sage: adic_seq(972,1001,10,15)
        (972, 1001, [2, 7, 9, 8, 2, 0, 1, 7, 9, 8, 2, 0, 1, 7, 9])
        sage: adic_seq(972,1001,10,20)
        (972, 1001, [2, 7, 9, 8, 2, 0, 1, 7, 9, 8, 2, 0, 1, 7, 9, 8, 2, 0, 1, 7])

    """
    b = []
    b_seq = []
    G = gcd(u,q)
    u = ZZ(u/G)
    q = ZZ(q/G)
    if gcd(q,x) != 1: #check if the denominator is relatively prime to x
        raise ValueError, "After reduction of u and q, x and q are not relatively prime"
    x = ZZ(x)
    N = int(N)
    for i in range(N):
        if i == 0:
            S = range(x)
        else:
            S = range(int(b_seq[i-1]),int(x^(i+1)),int(x^i))
        for j in S:
            if (u-q*j)%x^(i+1) == 0:
                if i == 0:
                    b.append(ZZ(j))
                else:
                    b.append(ZZ((j-b_seq[i-1])/x^(i)))
                b_seq.append(ZZ(j))
    return u,q,b


def xu(As, M):
    """
    This outputs the rational which "best fits" the
    decimal expansion in a.

    INPUT:
         As - a list of k integers in [0,M-1]
         M - a modulus (such as 10)

    EXAMPLES:
        sage: adic_seq(972,1001,10,20)
        (972, 1001, [2, 7, 9, 8, 2, 0, 1, 7, 9, 8, 2, 0, 1, 7, 9, 8, 2, 0, 1, 7])
        sage: xu([2, 7, 9, 8, 2, 0, 1, 7, 9, 8, 2], 10)
        (72849565132, 384693842281)
        sage: xu([2, 7, 9, 8, 2, 0, 1, 7, 9, 8, 2, 0, 1, 7, 9], 10)
```

```
        (10846647308, 62610590489)
        sage: xu([2, 7, 9, 8, 2, 0, 1, 7, 9, 8, 2, 0, 1, 7, 9, 8, 2, 0, 1, 7], 10)
        (972, 1001)
        sage: xu([0,1,0,0,0,0,0],10)
        (10, 1)
        sage: xu([0,0,1,0,0,0,0],10)
        (100, 1)
        sage: xu([1,0,0,0],10)
        (1, 1)
        sage: xu([1,0,0],10)
        (-9, 91)
        sage: xu([2, 7, 8, 2, 1, 7, 8, 2, 1, 7, 8, 2, 1],10)
        (72, 101)
        sage: xu([2, 7, 8, 2, 1, 7, 8, 2, 1],10)
        (149711144, 196122577)

    QUESTION: Do you need 3 times the period to arrive with the
    correct answer?

    """
    k = len(As)
    B = 3
    P = interpolation_set(M)
    a = 1+M*sum([As[j]*M**j for j in range(k)])
    m = 0
    h = [0 for i in range(k+1)]
    r = [0 for i in range(k+1)]
    h[0] = 0; r[0] = 1
    h[1] = 1+M*sum([As[j]*M**j for j in range(B)])
    r[1] = 1+M**B
    for i in range(1, k):
        #print "start", numerator((-1+(h[i]/r[i]))/M), denominator((-1+(h[i]/r[i]))/M)
        if (h[i]-a*r[i])%(M**(i+1)) <> 0:
            s = ifsinP(i,h,r,a,M,P)
            t = 0
            if s<>0:
                h[i+1] = s*h[i]
                r[i+1] = s*r[i]
                #print "type 1 update", i, numerator((-1+(h[i]/r[i]))/M),
denominator((-1+(h[i]/r[i]))/M)
            else:
                s,t = findst(h[i]-r[i]*a,M**(i-m)*(h[m]-r[m]*a),i,M)
                h[i+1] = s*h[i]+t*M**(i-m)*h[m]
                r[i+1] = s*r[i]+t*M**(i-m)*r[m]
                #print "type 2 update", i, numerator((-1+(h[i]/r[i]))/M),
  denominator((-1+(h[i]/r[i]))/M)
```

```
            cond1 = bool(height(h[i+1],r[i+1],M)>height(h[i],r[i],M))
            cond2 = bool(height(h[i],r[i],M) <= i-m+height(h[m],r[m],M))
            if (cond1 and cond2 and t<> 0):
                m = i
                #print "turn updating", i, numerator((-1+(h[i]/r[i]))/M),
   denominator((-1+(h[i]/r[i]))/M)
        else:
              h[i+1] = h[i]
              r[i+1] = r[i]
    u = numerator((-1+(h[k]/r[k]))/M)
    q = denominator((-1+(h[k]/r[k]))/M)
    t = maxpower(u,q,M)
    return u/M**t,q/M**t

#####################################################
```