A FASTER ALGORITHM FOR TESTING POLYNOMIAL REPRESENTABILITY OF FUNCTIONS OVER FINITE INTEGER RINGS

ASHWIN GUHA AND AMBEDKAR DUKKIPATI

Abstract. Given a function from $\mathbb{Z}_n$ to itself one can determine its polynomial representability by using Kempner function. In this paper we present an alternative characterization of polynomial functions over $\mathbb{Z}_n$ by constructing a generating set for the $\mathbb{Z}_n$-module of polynomial functions. This characterization results in an algorithm that is faster on average in deciding polynomial representability. We also extend the characterization to functions in several variables.

1. Introduction

In this paper we deal with the following question: given a function from a finite integer ring to itself does there exist a polynomial that evaluates to the function? In the case of real numbers $\mathbb{R}$, if the function is specified at only a finite number of points it is possible to obtain a polynomial using Lagrange interpolation [11]. For analytic functions one may get an approximation using Taylor's series. This problem has been well-studied over finite fields as well. It was noted by Hermite [7] that every function over finite field of the form $\mathbb{Z}_p$, which is the set of integers modulo prime p, can be represented by a polynomial. This result was extended by Dickson [5] to finite fields F_q, where q is a prime power. Moreover, it was also shown that there exists a unique polynomial of degree less than q that evaluates to the given function. A thorough study of finite fields can be found in [12].

The property of polynomial representability does not hold over finite commutative rings. In this paper we study the problem of polynomial representability over finite integer rings $\mathbb{Z}_n$, which is the set of residue classes of $\mathbb{Z}$ modulo n.

The earliest work in this direction was by Kempner [10]. It was proved that the only residue class rings over which all functions can be represented by polynomials are $\mathbb{Z}_p$, where p is prime. Kempner [10] also introduced the function (sometimes referred to as Smarandache function) defined as follows.

Definition 1.1. Kempner function $\mu : \mathbb{N} \longrightarrow \mathbb{N}$ is defined as $\mu(n)$ is the smallest positive integer such that $n \mid \mu(n)!$.

The Kempner function plays an important role in the study of polynomial functions. In his work, Kempner showed that there exists a polynomial of degree less than $\mu(n)$ that evaluates to a function over $\mathbb{Z}_n$, if the function is polynomially representable. An easy method to calculate $\mu(n)$ is also given in [10]. One can show that when n factors into primes as $p_1^{e_1} p_2^{e_2} \ldots p_t^{e_t}$, then $\mu(n) = \max(\mu(p_i^{e_i}))$ is of the form

$r \cdot p_k$ for some prime divisor p_k of n where r is a positive integer less than or equal to e_k.

It is obvious that the Kempner function is not monotonic: when n is prime $\mu(n) = n$, otherwise $\mu(n) < n$. Kempner function has been studied for its own merit and a discussion on the properties of this function is beyond the scope of this paper. However, one may claim that as n increases, $\mu(n)$ tends to be much smaller than n, by which one means that for most cases $\mu(n)$ tends to be sub-logarithmic compared to n [13].

Polynomial representation in $\mathbb{Z}_n$ has since then been studied by Carlitz [2]. The number of polynomial functions over $\mathbb{Z}_n$, when n is a prime power, is given by Keller and Olson [9]. This was extended to arbitrary positive integer n by Singmaster [15], where the Kempner function was used to give a canonical representation for the polynomial functions. Other notable results are given in [14, 1, 3, 4].

Recently, the problem of polynomial representability of functions in several variables has been studied by Hungerbühler and Specker [8]. In this work, an elegant characterization of polynomial functions was given by generalizing the Kempner function to several variables. The result makes use of partial difference operator to determine whether a given function from $\mathbb{Z}_n^m$ to $\mathbb{Z}_n$ is polynomially representable. This work does not provide a computational complexity analysis but one can see that this method does not lead to an efficient algorithm for verifying polynomial representability of the functions. The characterization involves repeated computation of the difference operator leading to an algorithm whose time complexity is very large. In terms of computation, its performance is comparable to the intuitive method of checking for existence of scalars $c_0, \ldots, c_{\mu(n)-1} \in \mathbb{Z}_n$ such that the polynomial $\sum_{i=0}^{\mu(n)-1} c_i X^i$ evaluates to the given function. For instance, in the case of single variable, the computation of $\Delta^k g(0)$ requires $O(k)$ operations for each $0 \leq k \leq n$, hence checking polynomial representability may require $O(n^2)$ operations.

In this paper we present a new characterization by adopting an entirely new approach that gives rise to a faster algorithm. For this, we generalize a characterization of polynomial functions over $\mathbb{Z}_{p^e}$ that is proposed in [6].

Contributions. In this paper we give an alternative characterization of polynomial functions over $\mathbb{Z}_n$. The new characterization is based on the fact that the set of polynomial functions forms a $\mathbb{Z}_n$-submodule of the $\mathbb{Z}_n$-module of all functions from $\mathbb{Z}_n$ to itself. We describe a 'special' generating set for this $\mathbb{Z}_n$-module of polynomial functions. When n is prime this generating set forms the standard basis for the vector space of polynomial functions. We present a new algorithm based on this characterization and show that this is faster on average in deciding the polynomial representability of functions. We also extend the characterization to functions in several variables and present a analysis of the algorithm in this case.

Organization. The paper is organized as follows. Section 2 contains the notation and necessary basic lemmas. The main theorem and the characterization are given in Section 3. An algorithm based on the result is given in Section 4. In Section 5 we discuss the complexity of the algorithm and compare its performance against an algorithm that makes use of canonical set of generators. The result is extended to functions in several variables in Section 6. Section 7 contains the concluding remarks.

2. Background

Throughout this paper we use n to denote a positive integer of the form $n = p_1^{e_1} p_2^{e_2} \dots p_t^{e_t}$, where $p_1 < p_2 < \dots < p_t$ are distinct primes. Kempner function is denoted by μ (Definition 1.1). Since n is fixed through out this paper, we abbreviate $\mu(n)$ to μ in some formulae. In $\mathbb{Z}_n$, each element of the congruence class is represented by the least non-negative residue modulo n and all computations are performed modulo n unless explicitly mentioned otherwise. Polynomials are of the form $c_0 + c_1 X + \dots + c_r X^r$, where X is the indeterminate and coefficients are from $\mathbb{Z}_n$.

A function $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ is represented as an n-tuple $(a_0, a_1, \dots, a_{n-1})$, where the i^{th} component $a_i = f(i)$, for $i = 0, \dots, n-1$. Hence we denote the set of all functions by $\mathbb{Z}_n^n$.

Given $v = (a_0, a_1, \dots, a_{n-1})$, $v^{<k>}$ represents the k^{th} cyclic shift to the right, for $k = 0, \dots, n-1$. That is

$$v^{<k>} = (a_{n-k}, a_{n-k+1}, \dots, a_{n-k-1}),$$

and we assume that $v^{<0>} = v$. In other words $v^{<k>}(i) = v(i-k)$ for all $k = 0, \dots, n-1$.

Given a set $\{v_1, v_2, \dots, v_r\} \subset \mathbb{Z}_n^n$, $\langle v_1, v_2, \dots, v_r \rangle$ denotes the $\mathbb{Z}_n$-module generated by that set. $\langle\langle v_1, v_2, \dots, v_r \rangle\rangle$ denotes the $\mathbb{Z}_n$-module generated by v_i with $i = 1, \dots, r$ along with their cyclic shifts, *i.e.*, $\langle\langle v_1, v_2, \dots, v_r \rangle\rangle = \{\sum \alpha_{ij} v_i^{<j>} \mid \alpha_{ij} \in \mathbb{Z}_n, i = 1, \dots, r,\ j = 0, \dots, n-1\}$. We say a function is polynomial if there exists some polynomial in $\mathbb{Z}_n[X]$ that evaluates to the given function.

We now make a few simple observations that are easy to verify.

Lemma 2.1. *Suppose $v \in \mathbb{Z}_n^n$ is a polynomial function. Then $v^{<k>}$ is also a polynomial function for all $k = 0, \dots, n-1$.*

This is easy to see since if $f(X) \in \mathbb{Z}_n[X]$ evaluates to v, then $f(X-k)$ which is also a polynomial evaluates to $v^{<k>}$.

Lemma 2.2. *Suppose $u, v \in \mathbb{Z}_n^n$ are polynomial functions. Then $\alpha u + \beta v$ is also a polynomial function for all $\alpha, \beta \in \mathbb{Z}_n$. In other words, the set of all polynomial functions forms a $\mathbb{Z}_n$-module.*

This is also obvious since if $f(X)$ and $g(X) \in \mathbb{Z}_n[X]$ evaluate to u and v respectively then $\alpha f + \beta g$ is also polynomial that evaluates to $\alpha u + \beta v$.

Lemma 2.3. *Suppose $u, v \in \mathbb{Z}_n^n$ are polynomial functions. Then $u \cdot v$ defined by componentwise multiplication,*

$$(u \cdot v)(x) = u(x) \cdot v(x), \text{ for all } x \in \mathbb{Z}_n$$

is also a polynomial function.

This is simply the assertion that if $f(X)$ and $g(X)$ are polynomials then $f(X)g(X)$ is also a polynomial. This lemma states that the polynomial functions form a $\mathbb{Z}_n$-algebra. Our objective is to provide a set of generators that generate the set of polynomial functions as a $\mathbb{Z}_n$-module. In particular we look for a set S such that $\langle\langle v \mid v \in S\rangle\rangle$ is the set of polynomial functions.

Definition 2.4. We call a set $S \subset \mathbb{Z}_n^n$, $\mathbb{Z}_n$-multiplicatively-closed if for all $u, v \in S$, $u \cdot v = \alpha w$ for some $\alpha \in \mathbb{Z}_n$, $w \in S$, where α may be zero.

This definition is similar to that of closure for any binary operation except that we allow the product to be a scalar multiple of an element in the set. Such a definition ensures that the module generated by such a set S is equal to the algebra generated by S. We now state the first non-trivial yet simple lemma.

Lemma 2.5. *Let $S \subset \mathbb{Z}_n^n$ be a $\mathbb{Z}_n$-multiplicatively-closed set. If the functions corresponding to 1 and X belong to the module generated by S, then every polynomial function belongs to the module generated by S.*

Proof. Let the functions induced by polynomials 1 and X belong to $\langle S\rangle$. This means the vectors $(1, 1, \ldots, 1)$ and $(0, 1, 2, \ldots, n-1)$ lie in $\langle S\rangle$. It suffices to show that evaluations of $1, X, X^2, \ldots, X^{\mu-1}$ lie in $\langle S\rangle$, since any polynomial function can be represented by a unique polynomial of degree less than μ. Let $X = \sum_{i=1}^{k} a_i u_i$, where $a_i \in \mathbb{Z}_n, u_i \in S$.

$$X^2 = X \cdot X = (\sum_{i=1}^{k} a_i u_i) \cdot (\sum_{i=1}^{k} a_i u_i) = \sum_{1\le i,j\le k} a_i a_j u_i \cdot u_j = \sum_{i=1}^{l} c_i v_i,$$

where $c_i \in \mathbb{Z}_n, v_i \in S$. Hence, function induced by X^2 belongs to $\langle S\rangle$. Similarly one can show that all exponents of X lie in $\langle S\rangle$. □

3. Characterization

Let $n = p_1^{e_1} p_2^{e_2} \ldots p_t^{e_t}$. Consider the functions $u_{p_i,j} : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ defined as follows. [1]

$$u_{p_i,j}(a) = \begin{cases} \frac{n}{p_i^{e_i}} a^j & \text{if } p_i \mid a, \\ 0 & \text{if } p_i \nmid a, \end{cases} \tag{1}$$

for all $i = 1, \ldots, t$, $j = 0, \ldots, e_i - 1$. In vector notation, when $j \neq 0$,

$$u_{p_i,j} = \frac{n}{p_i^{e_i}}(0, \ldots, (p_i)^j, \ldots, (2p_i)^j, \ldots, (n-p_i)^j, 0, \ldots, 0),$$

where entries for non-multiples of p_i are zero. The cyclic shifts of $u_{p_i,j}$ are defined as

[1]In the function definition, a is the canonical representative of a congruence class. $p_i \mid a$ means p_i divides a as integers.

$$u_{p_i,j}^{<k>}(a) = \begin{cases} \frac{n}{p_i^{e_i}}(a-k)^j & \text{if } a \equiv k \pmod{p_i}, \\ 0 & \text{otherwise,} \end{cases}$$

which corresponds to $u_{p_i,j}$ shifted by k places to the right for $i = 1, \ldots, t, j = 0, \ldots, e_i - 1$. Cyclic shifts of the form $u_{p_i,j}^{<k>}$ when k is a multiple of p_i can be written as a linear combination of elements in $\{u_{p_i,\ell} \mid \ell = 0, 1, \ldots, j\}$. Hence we only need to consider the first p_i shifts for each prime. We now show that $u_{p_i,j}$ along with their cyclic shifts form a generating set for the module of polynomial functions.

Lemma 3.1. *$u_{p_i,j}$ defined in (1) is a polynomial function for $i = 1, \ldots, t, j = 0, \ldots, e_i - 1$.*

Proof. It suffices to provide a polynomial that evaluates to each of the functions. For fixed $i \in \{1, \ldots t\}$ and $j \in \{0, \ldots, e_i - 1\}$ we give a polynomial that evaluates to $u_{p_i,j}$. Consider the monomial $X^{\phi(n)}$, where $\phi(n)$ is Euler's totient function. Since $\phi(n) \geq e_i$ for $n > 1$, $a^{\phi(n)} \equiv 0 \pmod{p_i^{e_i}}$ if $p_i \mid a$. If $p_i \nmid a$, p_i and a are relatively prime and $a^{\phi(n)} \equiv 1 \pmod{p_i^{e_i}}$ by Euler's theorem. Hence for all $a \in \mathbb{Z}_n$ we have

$$a^{\phi(n)} = \begin{cases} 1 \pmod{p_i^{e_i}} & \text{if } p_i \nmid a \\ 0 \pmod{p_i^{e_i}} & \text{if } p_i \mid a. \end{cases}$$

Then the polynomial $1 - X^{\phi(n)} \equiv (n-1)X^{\phi(n)} + 1$ corresponds to function

$$(1 - X^{\phi(n)})(a) = \begin{cases} 1 \pmod{p_i^{e_i}} & \text{if } p_i \mid a \\ 0 \pmod{p_i^{e_i}} & \text{if } p_i \nmid a \end{cases}$$

and the polynomial $X^j(1 - X^{\phi(n)})$ corresponds to the function

$$X^j(1 - X^{\phi(n)})(a) = \begin{cases} a^j \pmod{p_i^{e_i}} & \text{if } p_i \mid a \\ 0 \pmod{p_i^{e_i}} & \text{if } p_i \nmid a \end{cases}$$

for $j = 0, \ldots, e_i - 1$.

Since $\frac{n}{p_i^{e_i}}$ and $p_i^{e_i}$ are relatively prime we have

$$\frac{n}{p_i^{e_i}} X^j(1 - X^{\phi(p^n)})(a) = \begin{cases} \frac{n}{p_i^{e_i}} a^j & \text{if } p_i \mid a \\ 0 & \text{if } p_i \nmid a \end{cases}$$

which is the vector $u_{p_i,j}$ for $j = 0, \ldots, e_i - 1$. □

From Lemma 2.1 it follows that the cyclic shifts $u_{p_i,j}^{<k>}$ are also polynomial functions for $k = 0, \ldots, p_i - 1$.

Lemma 3.2. *$\{u_{p_i,j}^{<k>} | i = 1, \ldots, t, j = 0, \ldots, e_i - 1, k = 0, \ldots, p_i - 1\}$ is $\mathbb{Z}_n$-multiplicatively-closed.*

Proof. Case (i) : Consider $u_{p_i,j_1}^{<k_1>}$ and $u_{p_i,j_2}^{<k_2>}$ for a fixed i where $k_1 \neq k_2$ and j_1, j_2 are arbitrary.

$$(u_{p_i,j_1}^{<k_1>} \cdot u_{p_i,j_2}^{<k_2>})(a) = u_{p_i,j_1}^{<k_1>}(a) \cdot u_{p_i,j_2}^{<k_2>}(a) = 0$$

since at least one of the two will be zero.
Case (ii) : For a fixed i consider $u^{<k>}_{p_i,j_1}$ and $u^{<k>}_{p_i,j_2}$.

$$\begin{aligned}(u^{<k>}_{p_i,j_1}\cdot u^{<k>}_{p_i,j_2})(a) &= u^{<k>}_{p_i,j_1}(a)\cdot u^{<k>}_{p_i,j_2}(a)\\ &= \begin{cases}\frac{n}{p_i^{e_i}}(a-k)^{j_1}\cdot\frac{n}{p_i^{e_i}}(a-k)^{j_2} & \text{if } a\equiv k \pmod{p_i}\\ 0 & \text{otherwise}\end{cases}\\ &= \begin{cases}\frac{n}{p_i^{e_i}}\frac{n}{p_i^{e_i}}(a-k)^{j_1+j_2} & \text{if } a\equiv k \pmod{p_i}\\ 0 & \text{otherwise}\end{cases}\\ &= \frac{n}{p_i^{e_i}}u^{<k>}_{p_i,j_1+j_2}.\end{aligned}$$

Note that if $j_1+j_2\geq e_i$ then this corresponds to the zero function.
Case (iii) : Consider distinct p_{i_1} and p_{i_2} with arbitrary j_1,j_2. We need not consider cyclic shifts here since they are essentially same.

$$\begin{aligned}(u_{p_{i_1},j_1}\cdot u_{p_{i_2},j_2})(a) &= u_{p_{i_1},j_1}(a)\cdot u_{p_{i_2},j_2}(a)\\ &= \begin{cases}\frac{n}{p_{i_1}^{e_{i_1}}}a^{j_1}\cdot\frac{n}{p_{i_2}^{e_{i_2}}}a^{j_2} & \text{if } p_{i_1}p_{i_2}\mid a\\ 0 & \text{otherwise.}\end{cases}\end{aligned}$$

But $n\mid \frac{n}{p_{i_1}^{e_{i_1}}}\frac{n}{p_{i_2}^{e_{i_2}}}$, hence it is the zero function. □

Consider the sum of first p_1 shifts of $u_{p_1,0}$, $\sum_{k=0}^{p_1-1}u^{<k>}_{p_1,0}$. This corresponds to the constant function $\frac{n}{p_1^{e_1}}(1,1,\ldots,1)$. One can similarly obtain the functions $\frac{n}{p_i^{e_i}}(1,1,\ldots,1)$ for $i=2,\ldots,t$. We know that as integers

$$\gcd(\frac{n}{p_1^{e_1}},\frac{n}{p_2^{e_2}},\ldots,\frac{n}{p_t^{e_t}})=1.$$

From Bezout's lemma there exist $a_1,a_2,\ldots,a_t\in\mathbb{Z}$ such that

$$1=a_1\frac{n}{p_1^{e_1}}+a_2\frac{n}{p_2^{e_2}}+\ldots+a_t\frac{n}{p_t^{e_t}}.$$

The above statement is true considering $\frac{n}{p_i^{e_i}}, i=1,\ldots,t$ as integers. The statement is equally valid if we were to consider the corresponding equivalence classes modulo n. Therefore we have:

$$(1,1,\ldots,1)=\sum_{i=1}^{t}a_i\frac{n}{p_i^{e_i}}(1,1,\ldots,1)=\sum_{i=1}^{t}\sum_{k=0}^{p_i-1}a_iu^{<k>}_{p_i,0}.$$

This means the vector corresponding to the constant polynomial 1 can be written as a linear combination of elements in $\{u^{<k>}_{p_i,0}\mid i=1,\ldots,t,k=0,\ldots,p_i-1\}$, *i.e.*, $1\in\langle\langle u_{p_i,0}\mid i=1,\ldots,t\rangle\rangle$. We will employ a similar method to show that vector corresponding to polynomial X, *i.e.*, $(0,1,2,\ldots,n-1)$ belongs to the module $\langle\langle u_{p_i,j}\mid i=1,\ldots,t,j=0,\ldots,e_i-1\rangle\rangle$.

Lemma 3.3. *Function induced by the polynomial X can be written as a linear combination of elements in $\langle\langle u_{p_i,j} \mid i = 1, \dots, t, j = 0, \dots, e_i - 1\rangle\rangle$.*

Proof. For a fixed i consider the evaluation of $\frac{n}{p_i^{e_i}} X$.

$$\begin{aligned}
\frac{n}{p_i^{e_i}} X &= \frac{n}{p_i^{e_i}}(0, 1, 2, \dots, n-1) \\
&= \frac{n}{p_i^{e_i}}(0, 0, 0, \dots, p_i, 0, \dots, 0, 2p_i, \dots, 3p_i, \dots, 0) \\
&\quad + \frac{n}{p_i^{e_i}}(0, 1, 0, \dots, 0, p_i + 1, \dots, 2p_i + 1, \dots, 3p_i + 1, \dots, n - p_i + 1, \dots, 0) \\
&\quad + \frac{n}{p_i^{e_i}}(0, 0, 2, \dots, 0, p_i + 2, \dots, 2p_i + 2, \dots, 3p_i + 2, \dots, n - p_i + 2, \dots, 0) \\
&\qquad \vdots \\
&\quad + \frac{n}{p_i^{e_i}}(0, \dots, 0, p_i - 1, \dots, 2p_i - 1, \dots, 3p_i - 1, \dots, n - 1) \\
&= u_{p_i,1} \\
&\quad + u_{p_i,1}^{<1>} + 1u_{p_i,0}^{<1>} \\
&\quad + u_{p_i,1}^{<2>} + 2u_{p_i,0}^{<2>} \\
&\qquad \vdots \\
&\quad + u_{p_i,1}^{<p_i-1>} + (p_i - 1)u_{p_i,0}^{<p_i-1>}.
\end{aligned}$$

Since we know that monomial X can be represented as a linear combination of $\frac{n}{p_i^{e_i}} X$, it effectively means the function evaluated by polynomial X belongs to the module generated by $u_{p_i,0}^{<k>}$ and $u_{p_i,1}^{<k>}$. □

Now that we have functions of $1, X \in \langle\langle u_{p_i,j}\rangle\rangle$, Lemma 2.5 directly gives us the following theorem.

Theorem 3.4. $\{u_{p_i,j}^{<k>} \mid i = 1, \dots, t, j = 0, \dots, e_i - 1, k = 0, \dots, p_i - 1\}$ *generates the $\mathbb{Z}_n$-module of polynomial function from $\mathbb{Z}_n$ to itself.*

Written explicitly the generators are as follows.

$$\begin{aligned}
u_{p_1,0} &= \frac{n}{p_1^{e_1}}(1,0,\dots,0,1,\dots,1,\dots,0)\\
u_{p_1,1} &= \frac{n}{p_1^{e_1}}(0,0,\dots,0,p_1,\dots,2p_1,\dots,0)\\
&\vdots\\
u_{p_1,e_1-1} &= \frac{n}{p_1^{e_1}}(0,0,\dots,0,p_1^{e_1-1},\dots,(2p_1)^{e_1-1},\dots,0)\\
&\vdots\\
u_{p_t,e_t-1} &= \frac{n}{p_t^{e_t}}(0,0,\dots,0,p_t^{e_t-1},\dots,(2p_t)^{e_t-1},\dots,0).
\end{aligned}$$

When n is prime, $\mathbb{Z}_n$ is a field and the set of polynomial functions form a vector space over this field. The standard basis of the vector space is precisely the cyclic shifts of

$$u_{n,0} = (1,\underbrace{0,\dots,0}_{n\text{-1 times}})$$

as mentioned in Section 1.

For the case when n is a prime power of the form p^e the generators are precisely those given in [6].

$$\begin{aligned}
u_{p,0} &= (1,\underbrace{0,\dots,0}_{p-1\text{ times}},1,\underbrace{0,\dots,0}_{p-1\text{ times}},1,\dots,0)\\
u_{p,1} &= (0,\underbrace{0,\dots,0}_{p-1\text{ times}},p,\underbrace{0,\dots,0}_{p-1\text{ times}},2p,\dots,0)\\
u_{p,2} &= (0,\underbrace{0,\dots,0}_{p-1\text{ times}},p^2,\underbrace{0,\dots,0}_{p-1\text{ times}},(2p)^2,\dots,0)\\
&\vdots\\
u_{p,e-1} &= (0,\underbrace{0,\dots,0}_{p-1\text{ times}},p^{e-1},\underbrace{0,\dots,0}_{p-1\text{ times}},(2p)^{e-1},\dots,0).
\end{aligned}$$

When $n = p_1p_2\dots p_t$ the generators are cyclic shifts of

$$\begin{aligned}
u_{p_1,0} &= \frac{n}{p_1}(1,\underbrace{0,\dots,0}_{p_1-1\text{ times}},1,\underbrace{0,\dots,0}_{p_1-1\text{ times}},1,\dots,0)\\
u_{p_2,0} &= \frac{n}{p_2}(1,\underbrace{0,\dots,0}_{p_2-1\text{ times}},1,\underbrace{0,\dots,0}_{p_2-1\text{ times}},1,\dots,0)\\
&\vdots\\
u_{p_t,0} &= \frac{n}{p_t}(1,\underbrace{0,\dots,0}_{p_t-1\text{ times}},1,\underbrace{0,\dots,0}_{p_t-1\text{ times}},1,\dots,0).
\end{aligned}$$

Example 3.5. Consider $n = 12 = 2^2 \cdot 3$. The generators are

$$\begin{aligned} u_{2,0} &= (3,0,3,0,3,0,3,0,3,0,3,0) \\ u_{2,1} &= (0,0,6,0,0,0,6,0,0,0,6,0) \\ u_{3,0} &= (4,0,0,4,0,0,4,0,0,4,0,0) \end{aligned}$$

and their cyclic shifts.

4. Proposed Algorithm

We present an algorithm based on Theorem 3.4 in Algorithm 4.1. Let N be the number of generators given by Theorem 3.4. For each prime p_i, there are e_i generators and p_i cyclic shifts for each of them, which gives

$$N = p_1 e_1 + p_2 e_2 + \ldots + p_t e_t. \tag{2}$$

Let $u_0, u_1, \ldots, u_N$ be the generators and the input function $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ be of the form $(b_0, b_1, \ldots, b_{n-1}) \in \mathbb{Z}_n^n$. The key idea is to check if the given function is a linear combination of these generators. This is equivalent to checking if the following system of linear equations in variables $y_1, \ldots, y_N$ has a solution in $\mathbb{Z}_n$.

$$A \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix} \tag{3}$$

where A is an $n \times N$ matrix whose columns are the vectors $u_1, u_2, \ldots, u_N$. Since $N < n$, we have an over-defined system of equations. It is efficient to truncate the $n \times N$ matrix A and solve for the first N rows using Gaussian elimination to determine if a solution exists. Let B be the $N \times N$ matrix which consists of the first N rows of A in (3). We now solve for the smaller matrix

$$B \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{N-1} \end{pmatrix} \tag{4}$$

which is equivalent to checking for the first N components of f and then verify if the solution holds for remaining components of f.

Proof of correctness. Step 1 of Algorithm 4.1 checks for a necessary congruence condition every polynomial must satisfy. Step 1 checks if

$$g(a) \equiv g(a + p_i \ell) \pmod{p_i} \tag{5}$$

for all $p_i \mid n$ and $a, \ell \in \mathbb{Z}_n$. This step is useful in identifying a significant fraction of non-polynomial functions. Assuming that each entry in the n-tuple is arbitrary, for any prime factor p_i, number of functions that satisfy (5) is $n^{p_i} \left(\frac{n}{p_i}\right)^{n-p_i}$. In other words the fraction of functions that satisfy the condition is $\frac{1}{p_i^{n-p_i}}$ for each p_i, where $i = 1, \ldots, t$. Since the number of polynomial functions is much smaller than

Algorithm 4.1 Determination of Polynomial Functions

```
Input: $f = (b_0, b_1, \dots, b_{n-1})$, where $n = p_1^{e_1} \dots p_t^{e_t}$.

for $i = 1, \dots, t$ do                                                    ▷ Step 1
    for $j = 0, \dots, p_i - 1$ do
        for $\ell = 1, \dots, \frac{n}{p_i} - 1$ do
            if $b_j \not\equiv b_{j+\ell p_i} \pmod{p_i}$ then
                Output: $f$ is not polynomial.
                exit
                                                                            ▷ Step 2
if $B \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} b_0 \\ \vdots \\ b_{N-1} \end{pmatrix}$ has no solution then     ▷ B as in (4)
    Output: $f$ is not polynomial.
    exit
Let $(d_1, d_2, \dots, d_N)$ be the solution.
for $j = N, \dots, n - 1$ do                                                ▷ Step 3
    if $b_j \neq (\sum_{i=1}^{N} d_i u_i)(j)$ then
        Output: $f$ is not polynomial.
        exit
    else
        Output: $f$ is polynomial.
```

the number of non-polynomial functions, for most input functions the algorithm terminates at this step.

Step 2 computes the solution of (4). One must bear in mind that all computations are performed modulo n where division by p_i is not defined for $i = 1, \dots, t$. This means that whenever we encounter a case where division by p_i occurs it immediately implies that no solution exists in $\mathbb{Z}_n$, and the function f is not polynomial.

Suppose a solution exists, say, $(d_1, d_2, \dots, d_N)$. Step 3 checks if the solution holds for all components, *i.e.*, if

$$f = d_1 u_1 + d_2 u_2 + \dots d_N u_N.$$

The following example will help us understand the correctness of the algorithm.

Example 4.1. Consider $n = 12$. Let $f = (0, 1, 4, 9, 4, 1, 0, 1, 4, 9, 4, 1)$.

The generators are given in Example 3.5. Step 1 checks if

$$f(x + 2) \equiv f(x) \pmod 2,$$

$$f(x + 3) \equiv f(x) \pmod 3,$$

which is satisfied by f. Step 2 checks if the following system has a solution.

$$\begin{pmatrix} 4 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 4 & 3 & 0 & 6 & 0 \\ 4 & 0 & 0 & 0 & 3 & 0 & 6 \\ 0 & 4 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 3 & 0 & 0 \\ 4 & 0 & 0 & 3 & 0 & 6 & 0 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 4 \\ 9 \\ 4 \\ 1 \\ 0 \end{pmatrix}$$

A solution exists, namely $(0, 1, 1, 0, 3, 0, 0)$. In step 3 we check if the solution satisfies for all components, which it does. Hence f is polynomial.

5. Analysis of Algorithm

In this section we discuss the computational aspects of the algorithm. We study the complexity of Algorithm 4.1 and compare the performance of the algorithm with one based on the canonical set of generators, which is the set of functions corresponding to the monomials $\{1, X, X^2, \ldots, X^{\mu-1}\}$, where μ is as defined earlier.

First let us consider the set of generators $\{1, X, X^2, \ldots, X^{\mu-1}\}$. Let $\Lambda = \max\limits_{1 \le i \le t}\{p_i e_i\}$. We have $\mu \le \Lambda$, *i.e.*, Λ is an upper bound of μ for a given n, since $p_i^{e_i}$ divides $(p_i e_i)!$ for $i = 1, \ldots, t$. In order to determine if the given function $f = (b_0, b_1, \ldots, b_{n-1})$ is polynomial we need to check if there exist scalars $c_0, c_1, \ldots, c_{\mu-1} \in \mathbb{Z}_n$ such that $c_0 + c_1 X + \ldots + c_{\mu-1} X^{\mu-1}$ evaluates to f. This is equivalent to determining if the following system of linear equations similar to (3) has a solution.

$$\begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & 1 & 1^2 & \ldots & 1^{\mu-1} \\ 1 & 2 & 2^2 & \ldots & 2^{\mu-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (n-1) & (n-1)^2 & \ldots & (n-1)^{\mu-1} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{\mu-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix} \tag{6}$$

It is simpler to consider the first μ rows like we do in Algorithm 4.1. This step involves $O(\mu^3)$ operations. Suppose a solution exists, say, $(c_0, c_1, \ldots, c_{\mu-1})$. We then check if $\sum_{j=0}^{\mu-1} c_j i^j = b_i$, for $i = \mu, \ldots, n-1$. Evaluating the polynomial $\sum_{j=0}^{\mu-1} c_j X^j$ at each i requires μ multiplications. To evaluate the polynomial at $(n-\mu)$ points takes $O(n\mu)$ steps. Comparing the polynomial with $(b_0, b_1, \ldots, b_{n-1})$ requires one pass which is n operations.

Hence the complexity is $O(\mu^3) + O(n\mu) + O(n)$. For small values of n the term μ^3 dominates. However, as n increases $\mu \ll n$ and the effective complexity is $O(n\mu)$. Depending on the factorization of n, at worst it can be $O(n\Lambda)$.

We claim that the new set of generators reduces the number of computations. Before we proceed to study the complexity of the algorithm based on new generators we must remember that the complexity does not depend solely on the size of the input n, but also on the factorization of n.

Our proposed algorithm contains steps identical to the method mentioned above. Let N be the number of generators from (2). Observe that $\Lambda \leq N \leq \Lambda t$.

Step 1 requires a single pass for each prime factor p_i of n and takes $O(nt)$, since each traversal of the input takes $O(n)$ and there are t such primes. This does not affect the overall complexity adversely. In practice, it makes the algorithm faster by identifying many non-polynomial functions early without resorting to matrix calculations.

Step 2 solves for (4). We have an $N \times N$ matrix B which is larger than the one considered in (6) at most by factor of t. An important feature of matrix B is that most of its entries are zero. We can show that each row of A in (3) contains at most $(e_1 + \ldots + e_t)$ non-zero entries. Hence, of the nN entries of A at most $n(e_1 + \ldots + e_t)$ are non-zero. Moreover, the non-zero values are distributed uniformly over A. In other words,

$$\begin{aligned}\text{Fraction of non-zero entries} &< \frac{n(e_1 + e_2 + \ldots + e_t)}{nN} \\ &< \frac{e_1 + e_2 + \ldots + e_t}{p_1 e_1 + p_2 e_2 + \ldots + p_t e_t},\end{aligned}$$

which is also the fraction of non-zero entries in matrix B. This step takes $O(N^3)$ to solve in the worst case. It is possible that the matrix may be solved faster if it is sufficiently sparse. Suppose $(d_1, \ldots, d_N)$ is a solution.

Step 3 checks if $\sum_{i=1}^{N} d_i u_i(j) = b_j$ for the remaining components $j = N, \ldots, n-1$. Since each row contains fewer non-zero entries this step takes at most $n(e_1 + e_2 + \ldots + e_t)$ multiplications compared to $n\mu$ earlier. We can determine if $\sum_{i=1}^{N} d_i u_i = f$ in a single pass that takes $O(n)$. The total complexity of the new algorithm is $O(nt + N^3 + n(e_1 + \ldots + e_t))$. Although the second term is comparable to $\Lambda^3 t^3$, for large values of n the algorithm with new generators requires fewer multiplications.

Step-by-step break up of the complexity is given below.

$$\begin{aligned}\mathrm{T}(n) &= O(nt) + O(N^3) + O(n(e_1 + \ldots + e_t)) \\ &= O(n(e_1 + \ldots + e_t))\end{aligned}$$

The total complexity varies in magnitude depending on the factorization of n. We list some special cases where the algorithm performs significantly better.
Case 1: $n = p$. Algorithm 4.1 takes $O(n)$. Note that in this case $N = n$, hence Step 2 takes $O(n^3)$. However, this computation is unnecessary, since we know a priori every function is polynomially representable.
Case 2: $n = p^e$. Algorithm 4.1 takes $O(ne)$.
Case 3: $n = p_1 p_2 \ldots p_t$. Algorithm 4.1 takes $O(nt)$.

Some examples are given below.

Example 5.1. Consider $n = 29 \cdot 37^3 \cdot 53$. Then $\mu = 111$, $N = 193$, $\sum e_i = 5$. This means the new algorithm solves for a matrix roughly twice as big. However, the number of multiplications in Step 3 of our algorithm is $5n$ compared to $111n$ that may be required with canonical generators.

Example 5.2. Consider $n = 97{\cdot}101$. Then $\mu = 101$, $N = 198$, $\sum e_i = 2$. Once again the system of equations is larger, but the multiplications required is comparable to $2n$ instead of $101n$.

The above example represents the case where our algorithm performs much better, when the number of prime factors and the exponent of each prime is small. Note that the matrix required in Step 2 of Algorithm 4.1 is highly sparse, containing only two entries per row. This drastically reduces the calculation to solve the system of equations. The new algorithm performs poorly in the case when the exponents are much larger compared to the prime factors, in particular when the exponent e_i is much greater than p_i^2.

Example 5.3. Consider $n = 2^{15} \cdot 3^{10} \cdot 5^6$. Then $\mu = 25$, $N = 90$, $\sum e_i = 31$. The system of equations is larger yet the number of multiplications required turns out to be $31n$ rather than $25n$. Although in this case the number of multiplications is more, it still remains comparable to that using canonical generators.

One must note that in the above case μ was less than Λ, whereas in the previous examples the equality held. It is reasonable to assume that this equality holds for most values of n. Indeed, for a fixed positive integer n within a large enough upper bound, the possible values e_i may take is much less compared to those of p_i. Hence the new characterization is more efficient to check for polynomial representability for most values of n.

The next question that follows is determining the polynomial that evaluates to the given function. This is possible since we have the polynomials that correspond to the generators from Lemma 3.1 and the algorithm gives a suitable linear combination of generators. The polynomial thus obtained has a degree of $\phi(n)$. It is possible to get a lower degree polynomial by simply dividing it by $X(X-1)\dots(X-\mu+1)$. The remainder is of degree less than μ and evaluates to the same function. By similarly choosing suitable coefficients it is possible to arrive at the canonical representation mentioned in [15].

6. Polynomials in several variables

The set of generators described so far can be extended to multivariate functions in a natural way. Consider the set of functions in m variables $x_1, x_2, \dots, x_m$ over $\mathbb{Z}_n$. The intuitive set of generators to represent the polynomial functions are the evaluations of the monomials $X_1^{\alpha_1} X_2^{\alpha_2} \cdots X_m^{\alpha_m}$, where $\alpha_i = 0, \dots, \mu-1$, for $i = 1, \dots, m$, leading to μ^m generators. We wish to give a set of generators similar to that given in Theorem 3.4 for the set of polynomial functions from $\mathbb{Z}_n^m$ to $\mathbb{Z}_n$.

Proposition 6.1. *The module of polynomial functions in m-variables from $\mathbb{Z}_n^m$ to $\mathbb{Z}_n$ is generated by tensor product of vectors given for $\mathbb{Z}_n$ taken m at a time and their shifts,* i.e., *generators are given by $u_{p_{i_1},j_1} \otimes u_{p_{i_2},j_2} \otimes \dots \otimes u_{p_{i_m},j_m}$, where*

$$(u_{p_{i_1},j_1} \otimes \dots \otimes u_{p_{i_m},j_m})(a_1, \dots, a_m) = u_{p_{i_1},j_1}(a_1) \dots u_{p_{i_m},j_m}(a_m).$$

It must be noted that if $p_{i_1} \neq p_{i_2}$ (or any other pair), then the tensor product is simply zero. Effectively the generators are of the form $u_{p_i,j_1} \otimes u_{p_i,j_2} \otimes \ldots \otimes u_{p_i,j_m}$, where $i = 1, \ldots, t$. For a fixed p_i the number of generators, ignoring the shifts, is the number of solutions to the inequality $j_1 + j_2 + \ldots + j_m < e_i$ which is $\binom{m+e_i-1}{m}$. For each of these tensors there are p_i shifts along each of the m dimensions. Hence the number of generators corresponding to each p_i is $p_i^m \binom{m+e_i-1}{m}$. Summing up over all primes we get the total number of generators to be

$$N_m = p_1^m \binom{m+e_1-1}{m} + \ldots + p_t^m \binom{m+e_t-1}{m}.$$

Note that when we substitute $m = 1$, *i.e.*, the univariate case we get precisely $p_1 e_1 + \ldots + p_t e_t$ from (2) mentioned in Section 4.

Example 6.2. Consider the case of functions in two variables over $\mathbb{Z}_6$. The generators are

$$\begin{array}{c|cccccc}
 & 2 & 0 & 0 & 2 & 0 & 0 \\
\hline
2 & 4 & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 4 & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

$$\begin{array}{c|cccccc}
 & 3 & 0 & 3 & 0 & 3 & 0 \\
\hline
3 & 3 & 0 & 3 & 0 & 3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 3 & 0 & 3 & 0 & 3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 3 & 0 & 3 & 0 & 3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Note that $(2, 0, 0, 2, 0, 0) \otimes (3, 0, 3, 0, 3, 0)$ is a zero-matrix.

One may proceed to give an algorithm similar to Algorithm 4.1 for polynomials in several variables. Let $f : \mathbb{Z}_n^m \longrightarrow \mathbb{Z}_n$ be a function in m variables. We may now represent the function as an n^m-tuple $(b_0, b_1, \ldots, b_{n^m-1})$. Let $u_1, \ldots, u_{N_m}$ be the generators, each represented as an n^m-tuple. We check if the given function f is a linear combination of the generators, leading to a system of linear equations in variables $y_1, \ldots, y_{N_m}$, similar to (3) in Section 4.

$$A' \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N_m} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n^m-1} \end{pmatrix} \tag{7}$$

where A' is an $n^m \times N_m$ matrix whose columns are the generators $u_1, \ldots, u_{N^m}$. We solve for a smaller B' consisting of the first N_m rows of A' and then verify whether the solution holds for the remaining components.

$$B' \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N_m} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{N_m-1} \end{pmatrix} \tag{8}$$

A sketch of algorithm is given in Algorithm 6.1.

Algorithm 6.1 Determination of Polynomial Functions in several variables

```
                                                              ▷ Step 1
if $B' \begin{pmatrix} y_1 \\ \vdots \\ y_{N_m} \end{pmatrix} = \begin{pmatrix} b_0 \\ \vdots \\ b_{N_m-1} \end{pmatrix}$ has no solution then
    Output: $f$ is not polynomial.                            ▷ $B'$ as in (8)
    exit
Let $(d_1, d_2, \ldots, d_{N_m})$ be the solution.

for $j = N_m, \ldots, n^m - 1$ do                             ▷ Step 2
    if $b_j \neq (\sum_{i=1}^{N_m} d_i u_i)(j)$ then
        Output: $f$ is not polynomial.
        exit
    else
        Output: $f$ is polynomial.
```

As a preliminary step one can check if

$$f(a_1, \ldots, a_m) \equiv f(a_1 + \ell_1 p_i, \ldots, a_m + \ell_m p_i) \bmod p_i,$$

where $a_1, \ldots, a_m \in \{0, 1, \ldots, p_i - 1\}$ and $\ell_1, \ldots, \ell_m \in \{1, \ldots, \frac{n}{p_i} - 1\}$, for each p_i, as in Step 1 of Algorithm 4.1.

Note that in (7) the number of rows in A' is the size of input as in the case of single variable, but the matrix is much sparser. For each generator in $\{u_j \mid j = 1, \ldots, N_m\}$, if u_j corresponds to some prime p_i then the fraction of non-zero entries in u_j is at most $\frac{1}{p_i^m}$. Hence each prime p_i contributes $(\frac{n}{p_i})^m \binom{m+e_i-1}{m}$ non-zero entries to the matrix A'. For the matrix A' in (7)

$$\text{Fraction of non-zero entries} < \frac{n^m(\sum_{i=1}^{t}\binom{m+e_i-1}{m})}{n^m N_m}$$
$$< \frac{\sum_{i=1}^{t}\binom{m+e_i-1}{m}}{\sum_{i=1}^{t} p_i^m \binom{m+e_i-1}{m}}.$$

The uniform distribution of non-zero entries ensures that matrix B' in Step 1 of Algorithm 6.1 also has same sparsity. Assuming the system of linear equations is solved using Gaussian elimination, Step 1 of Algorithm 6.1 takes $O(N_m^3)$. Step 2 takes $O(n^m \sum_{i=1}^{t}\binom{m+e_i-1}{m})$ multiplications since each row of A' (and B') contains at most $\sum_{i=1}^{t}\binom{m+e_i-1}{m}$ non-zero entries. The total time complexity of Algorithm 6.1 is given by

$$T(n,m) = O(N_m^3) + O(n^m \sum_{i=1}^{t}\binom{m+e_i-1}{m}).$$

For most n and m, the values of N_m and $\sum_{i=1}^{t}\binom{m+e_i-1}{m}$ are smaller than n^m by several orders of magnitude. Hence the time complexity of the algorithm for functions in several variables is much less compared to the methods that result out of canonical generators or the characterization given in [8] .

7. Conclusion

In this paper we have provided an alternate characterization of polynomial functions over $\mathbb{Z}_n$ that results in improved algorithms for deciding polynomial representability. This characterization makes use of module structure of the set of polynomial functions and can be used to give a polynomial that evaluates to the polynomially representable function. By this, we can also arrive at the canonical representation mentioned in [15]. In addition, the characterization is also extended to polynomial functions in several variables.

Acknowledgments

Authors would like to thank anonymous reviewers for bringing our attention to the work of Hungerbühler and Specker (2006) and providing valuable suggestions that improved this paper immensely.

References

[1] J.V. Brawley and G.L. Mullen. Functions and polynomials over Galois rings. *Journal of Number Theory*, 41(2):156–166, 1992.

[2] L Carlitz. Functions and polynomials $(modp^n)$. *Acta arithmetica*, 9(1):67–78, 1964.

[3] Zhibo Chen. On polynomial functions from $\mathbb{Z}_n$ to $\mathbb{Z}_m$. *Discrete Mathematics*, 137(1):137–145, 1995.

[4] Zhibo Chen. On polynomial functions from $\mathbb{Z}_{n_1} \times \ldots \times \mathbb{Z}_{n_r}$ to $\mathbb{Z}_m$. *Discrete Mathematics*, 162(1):67–76, 1996.

[5] L.E. Dickson. The analytic representation of substitutions on a power of a prime number of letters with a discussion of the linear group. *The Annals of Mathematics*, 11(1/6):65–120, 1896.

[6] Ashwin Guha and Ambedkar Dukkipati. An algorithmic characterization of polynomial functions over $\mathbb{Z}_{p^n}$. *Algorithmica*, 10.1007/s00453-013-9799-7.

[7] C. Hermite. Sur les fonctions de sept lettres. *Comptes rendus de l'Acadmie des Sciences Paris*, 57:750–757, 1863.

[8] Norbert Hungerbühler and Ernst Specker. A generalization of the Smarandache function to several variables. *Integers: Electronic Journal of Combinatorial Number Theory*, 6(A23), 2006.

[9] G. Keller and FR Olson. Counting polynomial functions (mod p^n). *Duke Mathematical Journal*, 35(4):835–838, 1968.

[10] Aubrey J Kempner. Polynomials and their residue systems. *Transactions of the American Mathematical Society*, 22(2):240–266, 1921.

[11] IL Lagrange. Reflexions sur la resolution algebrique des equations.

[12] R. Lidl, H. Niederreiter, and P.M. Cohn. *Finite fields*, volume 20. Cambridge Univ Pr, 1997.

[13] Florian Luca. The average Smarandache function. In *Smarandache Notions (Proceedings of the Second International Conference on Smarandache Type Notions in Mathematics and Quantum Physics)*, volume 12, pages 19–27. Infinite Study, 2001.

[14] G. Mullen and H. Stevens. Polynomial functions (mod m). *Acta Mathematica Hungarica*, 44(3):237–241, 1984.

[15] David Singmaster. On polynomial functions (mod m). *Journal of Number Theory*, 6(5):345–352, 1974.

Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India.

E-mail address: `guha_ashwin@csa.iisc.ernet.in`

Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India.

E-mail address: `ad@csa.iisc.ernet.in`