Analysis of Permutation Polynomials over Finite Field as Verifiable Delay Function

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Anurag Jha
(111901010)

Boda Pranav Aditya
(111901018)

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

<u>CERTIFICATE</u>

*This is to certify that the work contained in the project entitled "**Analysis of Permutation Polynomials over Finite Field as Verifiable Delay Function**" is a bonafide work of **Anurag Jha** (**Roll No. 111901010**) and **Boda Pranav Aditya** (**Roll No. 111901018**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my guidance and that it has not been submitted elsewhere for a degree.*

Srimanta Bhattacharya

Assistant/Associate Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Acknowledgements

We would like to thank our professor Srimanta Bhattacharya for giving us the chance to work on this project, and we would want to express our gratitude.
We appreciate his advice and unwavering support, which have aided and encouraged us throughout the project.

Abstract

In this report, we first defined the fields before moving on to finite fields. Then, we defined permutation polynomials and offered numerous theorems and findings pertaining to these permutation polynomials. Additionally, we defined irreducible polynomials. After that, we went on to explain the various codes we had created for Sage Math, including those for generating permutation polynomials, listing out all binomial permutation polynomials, and listing out all binomial permutation polynomials such that f(x) + x is also a permutation polynomials.

Then we have explained what Variable Delay Functions are, and how we are planning to use these binomial permutation polynomials in Verifiable Delay Functions.

Contents

Chapter 1

Introduction

A field is a collection of elements that may perform addition and multiplication operations that are equivalent to those found in the real number system. **Finite fields** as the name suggests consists of finite number of elements. The cardinality of a field is called the order of field. The order of finite field is finite and are prime numbers or power of primes. i.e. order of finite field = p or p^k [where p is prime number and k is a positive integer]
One very common example of finite field is integer mod p.
The characteristic of field is defined as the number of times we need to add the copies to get 0, for a field of order p^k, this value will be p.

If we have a finite field of order q, a polynomial $f \in F_q[x]$ is called a **Permutation polynomial** of F_q if induced mapping $x \mapsto f(x)$ is a permutation of F_q.
$\forall$ permutation in field, there is a permutation function attached to it.

Chapter 2

Permutation Polynomials

Theorem: 1 *A polynomial f(x) = $\sum_{i=0}^{d} a_i x^i$ over finite field F is permutation polynomial if it induces a bijection from F to F.*

Theorem: 2 *Fermat's little theorem states that if p is a prime number, then for any integer a, the number a^p−a is an integer multiple of p.*
i.e. $a^p \equiv a \ mod \ p$

Fermat's little theorem gives one of the simplest permutation polynomials modulo a prime p: $f(x) = x^p$, satisfying that for all x belonging to integers, $f(x) \equiv$ x (mod p).

We observe that a polynomial $f \in F_q[x]$ is a permutation polynomial of F_q if and only if one of the following conditions holds: PP

1. The function f is onto.
2. The function f is one-to-one.
3. $\forall a \in F_q$, the equation $f(x) = a$ has solution in F_q
4. $\forall a \in F_q$, the equation $f(x) = a$ has a unique solution in F_q

Theorem: 3 *Every linear polynomial of the form $ax + b$ with over F_q having $a \neq 0$ is a permutation polynomial of F_q.*

Theorem: 4 *x^k is a permutation polynomial of F_q if and only if the $gcd(k, q-1) = 1$*

Theorem: 5 ***Irreducible Polynomial*** *is a polynomial that cannot be factored into product of two non-constant polynomials over the same field.*

$x^2 + 1$ is an example of a degree 2 polynomial over F_7. This is because if we consider the polynomial to be $(x + a)(x + b)$, we get $x^2 + (a + b)x + ab$, there are no two numbers a,b in mod 7 such that (ab) mod $7 == 1$ and $(a + b)$mod $7 == 0$.
We can use this polynomial to create extension field such as F_{49}.
$F_{49} = \frac{F_7[x]}{x^2+1}$
Here instead of $x^2 + 1$, we can use any other irreducible polynomial in F_7 as well.

$x^2 + 1$ is not an irreducible polynomial of degree 2 polynomial over F_2. This is because if we consider the polynomial to be $(x + 1)(x + 1)$, we get $x^2 + (1 + 1)x + 1$.
$x^2 + 2x + 1$ polynomial over F_2 is $x^2 + 0x + 1$.

Theorem: 6 *(**Hermite criterion**) [1] A polynomial $f(x) \in Fq[x]$ is a permutation polynomial of Fq if and only if following two conditions hold:*

1. *In Fq, f(x) has only 1 root ; and*
2. *for each integer t with $1 \leq t \leq q - 2$ and $t \neq$ 0 mod p, the reduction of ${f(x)}^t$ mod $(x^q - x)$ has degree $\leq$ q - 2.*

Let us try to prove that f has only 1 root in Fq if and only if $f(x)^{q-1}$ mod $(x^q - x) = \sum_{i=0}^{q-1} {b_i}^t x^i$, where ${b_{q-1}}^{(t)} = -\sum_{c \in Fq} {f(c)}^t$.
If f has only j roots in Fq, then:
${b_{q-1}}^{(q-1)} = -\sum_{c \in Fq} f(c)^{q-1} = -(q - j) =$ j.
Since $0 \leq j \leq q - 1$, we have:

${b_{q-1}}^{(q-1)} = 1$ if and only if $j = 1$

The Hermite criterion is a very powerful method to identify if a polynomial is permutation polynomial or not and we have implemented the same criterion in our codes as well.

The following can be concluded from the Hermite's criterion:

- $g(x)$ is obtained by the reduction of $f(x)^t$ mod $(x^q - x)$ and it belongs to $Fq[x]$, satisfying deg $g(x) \leq q - 1$ and $f(x)^t \equiv g(x)$ mod $(x^q - x)$.
- Since $c^q = c$, we have: $f(c)^t = g(c)$ for all $c \in Fq$.
- There is no permutation polynomials of Fq of degree d, if $d \geq 1$ is a divisor of $q - 1$.
- For any odd prime p, there is no permutation polynomial of over Zp of degree p $-$ 1.

Chapter 3

Sage codes

We have written numerous pieces of code that account for different permutation polynomial features and facets. Sage Documentation

3.1 Given a polynomial check if it is permutation polynomial or not

```

#This function uses cardinality to check if polynomial is PP or not
def permutationPolynomial_cardinalityCheck(f,n):
    output=set()
    for i in range(n):
        k=f(i)
        k=k%n
        output.add(k) # unique outputs are added to the set
    if len(output)==n :
        return 1
    return 0


```

```
#This function uses Hermite's criterion to check if polynomial is PP or
   not
def checkPermutationPolynomial(f,q):
    k=f.roots()
    if(len(k)!=1):  #Roots not equal to 1
        return 0
    for i in range(1,q-1):
        p=(f^i)%(x^q - x)
        if(p.degree() >= q-1):
            return 0    #if any degree is greater than equal to q-1
    return 1
```

3.2 Generate all the permutation polynomials of Fq where $q = 2^n$ and coefficients are permutation of q

```

x = PolynomialRing(RationalField(), 'x').gen()  #variable x
polynomials = []    #To store polynomials

def binaryToPolynomial(arr, n): #convert binary string to polynomial

    f = 0
    for i in range(0, n):
        f = f + arr[i]*(x^i)

    polynomials.append(f)   #append given polynomial to list

# Function to generate all binary strings
def generateAllBinaryStrings(n, arr, i):

    if i == n:
        nums.append(arr)
```

```python
        binaryToPolynomial(arr, n)
        return

    # assign "0" at ith position, and try for other permutations
    arr[i] = 0
    generateAllBinaryStrings(n, arr, i + 1)

    # assign "1" at ith position, and try for other permutations
    arr[i] = 1
    generateAllBinaryStrings(n, arr, i + 1)

#This function uses Hermite's criterion to check if polynomial is PP or not
def checkPermutationPolynomial(f,q):
    k=f.roots()
    if(len(k)!=1):  #Roots not equal to 1
        return 0
    for i in range(1,q-1):
        p=(f^i)%(x^q - x)
        if(p.degree() >= q-1):
            return 0    #if any degree is greater than equal to q-1
    return 1

# Driver Code
if __name__ == "__main__":

    n = 31
    arr = [None] * n

    #Generate binary strings
    generateAllBinaryStrings(n, arr, 0) #Generating binary

    for i in range(len(polynomials)):   #Print polynomials
        print(polynomials[i])

    countPP=0
    PPList=[]

    for i in range(len(polynomials)):
        if(checkPermutationPolynomial(polynomials[i],n)==1):    #If
   polynomial was PP
            countPP=countPP+1
            PPList.append(polynomials[i])

    print(countPP)      #Print number of permutation polynomials
    print(PPList)       #Print permutation polynomials
```

3.3 Find all permutation polynomials f(x) of form $x^r + ax^s$ where f(x) + x is also a permutation polynomial

Here: b = all values of F, r = all values of F,s = 0 to r

```

x = PolynomialRing(RationalField(), 'x').gen()  #variable x
polynomialsSet = set()  #set to store polynomials

def generatePolynomials(n):

    for r in range(n):
        for s in range(r):
            for b in range(n):
                f=x^r + b*(x^s) #generate all possible binomials
                polynomialsSet.add(f)   #append binomial to set

```

```
def checkPermutationPolynomial(f,q):
    k=f.roots()
    if(len(k)!=1):
        return 0
    for i in range(1,q-1):
        p=(f^i)%(x^q - x)
        if(p.degree() >= q-1):
            return 0
    return 1

# Driver Code
if __name__ == "__main__":

    n = 2^2
    arr = [None] * n

    generatePolynomials(n)

    countPP=0
    permutationPolynomialList=[]

    for p in polynomialsSet:
        f = p + x
        if(checkPermutationPolynomial(p,4)==1 and
   checkPermutationPolynomial(f,4)==1): #if both f(x) and f(x) + x is PP
            countPP=countPP+1
            permutationPolynomialList.append(p)

    print(countPP)  #Print number of permutation polynomials
    print(permutationPolynomialList)    #Print permutation polynomials
```

Chapter 4

Conclusion and Future Work

Verifiable Delay Function (VDF) is a function whose computation requires running sequential steps and the result can be efficiently verified . VDF
VDF is always guaranteed with short **clock time** i.e. Any observer can verify the output in a very short amount of time. No matter how many processors the task is divided it is still guaranteed it takes the same amount of clock time to complete.
We have managed to obtain all the binomial permutation polynomials for field with small order. For our future work we will try to increase the order of the field and try obtaining more binomial permutation polynomials. We will select the best polynomials that can be used in Verifiable Delay Function.

References

[1] C. J. Shallue, “Permutation polynomials of finite fields,” 2012. [Online]. Available: https://arxiv.org/abs/1211.6044