Galois Field Package Manual

Ryoh Fuji-Hara
University of Tuskuba

Abstract

When we implement an algebraic system like a group, a ring or a finite field within an existing symbolic computational language, then there is three possible approaches:
(1) define new operators: for example; define an operator %+% for addition on a finite field, then we could operate in the following fashion: 2 %+% 4 .
(2) change mode: for example; if we write ChangeAlgebra[GF,7], then every computation is done over GF(7) until we change to some other algebra.
(3) use the same operations within the existing system (i.e. +,-,*,/): for this we have to identify constants and variables to be in a certain algebra. In this method a lot of original functions of the system become available to the new algebra.

Due to the fact Mathematica is an object oriented language it facilitates the adaptation of the third method cited above. Therefore, the package allows us to use many functions of Mathematica over finite fields without any modification, i.e.; Solving linear equations, Inverse, Determinant, Derivations, Resultant, etc..

1. Finite Fields

A finite field (or Galois filed) is, roughly saying, a finite set having algebraic properties like Quotient, Real or Complex except topological and ordering properties. We define a finite field mathematically as follows:
Let K be a finite set. + and * are binary operations defined on K. If the following properties are satisfied then the an algebraic system (K,+,*) is called a *finite field* (or *Galois filed*).

1. + and * are close on K,
2. for any elements a and b of K, a+b=b+a and a*b=b*a,
3. for any elements a, b and c of K, (a+b)+c=a+(b+c) and (a*b)*c=a*(b*c),
4. for any elements a, b and c of K, a*(b+c)=a*b+a*c,
5. there exist unique additive identity 0 and multiplicative identity 1,
6. for any element a of K, there exists unique additive inverse -a. And for any nonzero element b of K, there exits unique multiplicative inverse b^{-1}.

If there are q elements in K, we write GF(q) for the finite field (K,+,*). Then the number of elements in K is called the *order* of the field.

Theorem 1. *There exists a finite field of order q if and only if q is a prime or prime power.*

If q is a prime number, then it is easy to construct a finite field of order q. Let K be {0,1, 2,..., p-1}. If we consider the addition and the multiplication on K to compute under the modulo p, $+_{(\text{mod } p)}$ and $*_{(\text{mod } p)}$, then this algebraic system (K, $+_{(\text{mod } p)}$, $*_{(\text{mod } p)}$) is the finite field GF(p), and it is called a *prime field*. If a subset S of the elements of a finite field F satisfies the above axioms with the same operators of F, then S is called a *subfield*.

Theorem 2. *Let F be a finite field of order q^n , p a prime number. There exists a subfield of order p^m if and only if m divides n.*

Consider a finite field with order of a prime power number. Let $f(x)=x^3+x+1$ be a polynomial over GF(2) which is unfactorable, such a polynomial is called a *irreducible polynomial*. Using the modulo x^3+x+1, we can reduce x^0 , x^1 , x^2 , ... to polynomials with degree less than 3.

Power	Polynomial	Vector	Regular(Decimal)
0	0	(000)	0
$x^0=1$	1	(100)	1
x^1	x	(010)	2
x^2	x^2	(001)	4
x^3	$1+x$	(110)	3
x^4	$x+x^2$	(011)	6
x^5	$1+x+x^2$	(111)	7
x^6	$1+\quad x^2$	(101)	5

The first and second columns are called *power* and *polynomial representations*, respectively. The third column is the triples of coefficients of the polynomial representations, called *vector representation*. The last column, called *regular representation*, is the integer representations of vectors which are regarded as binary numbers. The set of the above polynomials in the second column are close under the addition and the multiplication under the modulo f(x) and these operations on the set satisfy the axioms of finite field. This finite filed is said to be an *extension field* of degree 3 of GF(2) and write GF(2^3). Then the field GF(2) is called the *base filed* of GF(2^3). Note that it is not sufficient to generate all element of extension field 0, x^0, x^1,... are generated by an irreducible polynomial. If an irreducible polynomial generates all elements like the above then it is called a *primitive irreducible polynomial*.

Theorem 3. *For any prime or prime power q and any positive integer n, there exist a primitive irreducible polynomial of degree n over GF(q).*

We have now two finite fields F_1 and F_2 of same order q. If there is a 1-1 mapping $g: F_1 \to F_2$ such that g(x) + g(y) = g(x + y) , g(x)g(y) = g(xy), then the fields F_1 and F_2 are said to be *isomorphic*. If order of F_1 is less than order of F_2 then we say that F_1 is *homomorphic* to F_2.

Theorem 4. *Two finite fields with same order are isomorphic.*

The followings are important properties on finite fields:

Theorem 5. *For any element c of GF(q), $c^q=c$.*

Theorem 6. *For any nonzero element d of GF(q), $d^{q-1}=1$.*

The smallest positive integer n such that $\sum_{1\le i\le n} 1 = 0$ in GF(q) is called the *characteristic* of the field GF(q). The characteristic is a prime number for every finite field.

Theorem 7. *$(x+y)^p = x^p + y^p$ over a finite field of characteristic p.*

2 . Installation and Start-up

The "Galois Field Package" contains the following files:
GaloisField.m
GaloisFieldFactor.m
GF4
GF8
GF16
N

Just copy the all files into the folder “Applications” under Mathematica folder whichi is the Mathematica user setting folder. If you use MacOS X, the Mathematica user aetting folder is located under “Library” folder of user home directory. In the case of Windows, it is located under C:¥Documents and Settings

Now you can start Mathematica. In Mathematica session, if you type

```
In[1]:= <<GaloisField.m
GaloisField.m  was  loaded.
```

Then the Galois Field Package is ready to use. If you have the following message:

```
Mathematica can't find the file "GaloisField.m".
Please locate this file, or click "Cancel" if you don't want to open it.
```

You should correct the setting of $Path in the file "init.m".

3. Algebra Names and Constants

When we declare a finite field, we can name for the field with the following rules:

· The first character must be an alphabet.
· It is not allowed to use a symbol which is used in Mathematica.

· You will have a warning message if you use a name of one character .
It is called *algebra name*. We can use different algebra names for finite fields with the same order. Every element of a finite field, that is, constant, have the following form:

algebra name[integer]

The integer in the form is a regular representation of the finite field. For example, suppose we now have an algebra name K5 for GF(5), then we can write the following expressions:

```
In[3]:= K5[2] + K5[4]
Out[3]= K5[1]

In[3]:= K5[2]^3
Out[3]= K5[3]

In[5]:= K5[2] * K5[3] + K5[1]
Out[5]= K5[2]

In[7]:= K5[3]^-1
Out[7]= K5[2]

In[8]:= Sqrt[K5[4]]
Out[8]= K5[2]
```

In this case, the order of K5 is a prime, 5, therefore integers appear in [ ] are 0,1,...,4. In case of a extension field GF(q^n), the regular representation is used for integers in [ ]. The regular representation have some advantages as listed below:

- 0 is the additive identity and 1 is multiplicative identity for any field.
- In a finite field GF(q^n) with the base field GF(q), 0,1,,..., q-1 are always the elements of its base field.
- In a finite field with characteristic p, {0, 1,..., p-1} is always a subfield (prime field).
- Between an extension field of degree m and a extension field of degree n over GF(q), (m n), then there is the natural correspondence, $\varsigma : i \to i$, $0 \le i \le q^m - 1$, of homomorphism with respect to the additions.

Let K9 be the algebra name of GF(3^2). Then, for instance, K9[3] , K9[4], K9[5], K9[6] are representations of a, a+1, a+2, 2a, respectively.

```
In[12]:= K9[4] + K9[5]
Out[12]= K9[6]
```

```
In[13]:= K9[6]^-1
Out[13]= K9[7]

In[17]:= K9[3]^0
Out[17]= GF9[1]
```

K9[0] is the additive identity and K9[1] is the multiplicative identity of K9. K[0], K9[1], K[2] are the elements of the base filed GF(3) of K9. Algebra names are defined as *Listable* in Mathematica, therefore, it is easy to define a vector:

```
In[18]:= K9[{1,3,5}]
Out[18]= {K9[1],K9[3],K9[5]}
```

4. Types of Finite Fields

There are four types of declarations for finite fields in this package depending on constructions of finite fields.

I Prime fields
II Extension fields over a prime base field. (needs a Galois field object)
III Extension fields over a declared base field. (needs a Galois field object)
IV Extension fields over a declared base field. (no objects needed)

When we declare a finite field of type II or III, then there must exit the object of the finite field as a file, it is called a *Galois field object*. A Galois field object can be created as a pre-computation and saved as a file.

4.1 Type I

After loading the Galois Field Package by `<<GaloisField.m`, we declare a finite field by the function `DeclareGaloisField`.

```
In[1]:= DeclareGaloisField[K5,5]
  K5 was declared.
     Order is 5
     Characteristic is 5
```

This example declares a prime field of order 5, GF(5). Then we can use the algebra name K5 and constants K5[i] for i=0,1,...,4.

```
In[2]:= K5[3] + K5[4]
Out[2]= K5[2]
```

4.2 Type II and III

When you declare a type II or III finite field, just use the function DeclareGaloisField

```
In[2]:= DeclareGaloisField[GF16]
   GF16 was declared.
      Order is 16
      Characteristic is 2
      Irreducible Polynomial is {1, 1, 0, 0, 1}
      Base Field is GF2
      Extension Degree is 4
   GF2 was declared.
     Order is 2
     Characteristic is 2
```

If there does not exist the object GF16, you have the following message:

```
   Galois field object, GF16, does not exist.
```

Then you can make a type II object by the following function:

```
In[1]:=  MakeGaloisField[GF16,2,4,{1,1,0,0,1}]
```

The first argument is a name of object (also used as algebra name), the second is the order of the base field (must be a prime number), the third is the extention degree and the fourth argument is the coeficient list of a primitive irreducible polynomial over the base field. The function saves the object as a file with the name of the first argument.
When you make a type III object, The second argument of MakeGaloisField must be a algebra name of base field.

```
In[2]:=  MakeGaloisField[F16,GF4,2,{2,2,1}]
```

Of course, the base field must be declared before executing this function. Base field for type III is not necessary to be a prime field.

4.3 Type IV

Type IV Galois fields do not require objects for them. If a base field has been declared, then you can declare a type IV Galois field directory by the function DeclareGaloisField.

```
In[3]:= DeclareGaloisField[K16,GF4,2,{2,2,1}]
```

Since the fields of this type do not have any table for computations, in each computation on type IV finite field, elements are reformed to polynomials over the base field and compute under the modulo the irreducible polynomial. Therefore the irreducible polynomial is not necessary to be a primitive. Furthermore, if you use a reducible polynomial instead of irreducible, you can declare a polynomial ring.
Type IVGalois fields are not recommended for fields of order less than 1000 because of the computations are very slow.

5 Variables on Galois fields

The Galois Field Package has two kinds of variables over finite fields. One is a variable which represent an element of a field. When we see in a mathematics book "for an element x of the field K" , the symbol x represents an constant of K. This variable works like an element of K and may be substituted by a constant later. The second kind of variable is called an indeterminate or an algebraic variable. The algebraic variable never be substituted by a constant and dose not work like an element. Theorem 5 works on a variable which represents an element but not on an algebraic variable.

When we declare variables which represent elements of the finite field K9.

```
In[3]:= DeclareVariables[K9,x,y]
```

Since K9 is a finite field of order 9, the theorem 5 works like:

```
In[4]:= x^9
Out[4]= x
In[5]:= x^-10
```
Out[5]= x^{-2}

From the theorem 5, $x^9=x$ in GF(9). Exponents of variable x in outputs never be larger

than 8 (generally "order" - 1).
Mathematica does not print the form " x^0 " , Therefore Galois Field Package prints with the form " x^{Zero} ". The symbol Zero exactly works as numeric number 0.

```
In[5]:= x^0
Out[5]= x^Zero
In[6]:= % /. x-> K9[5]
Out[6]= K9[1]
```

The theorem 6 say that if x is an nonzero element of K9 then $x^8=1$. Galois Field Package have a declaration function for nonzero variable.

```
In[6]:= NonZero[x]
In[7]:= x^{0,1,2,3,4,5,6,7,8,9,10}
Out[7]= {GF9[1], x, x^2 , x^3, x^4, x^5, x^6, x^7, GF9[1], x, x^2 }
```

You can compare with ordinal variable y.

```
In[8]:= y^{0,1,2,3,4,5,6,7,8,9,10}
Out[8]= {y^Zero ,y , y^2, y^3 , y^4 , y^5 , y^6 , y^7 , y^8 , y , y^2 }

In[9]:= {x , y }^16
Out[9]= { K9[1], y^8 }
```

Note that 0-th power of 0 is indeterminate.

```
In[9]:= K9[0]^0
Power::K9[0]^-1 and K9[0]^0 are undefined
Out[9]= Indeterminate
```

Nonzero variables also can be declared by the NonZero option in the function DeclareVariables.

```
In[9]:= DeclareVariables[K9,x, NonZero -> True]
```

Next, When we declare an algebraic variable, we use the function DeclareAlgebraicVariavles or DeclareIndeterminates.

```
In[8]:= DeclareAlgebraicVariables[K9,X,Y]
```

Using the algebraic variable, we can define a polynomial ring GF(9)[X]. Since algebraic variables are not elements of a finite filed, the theorem 5 and 6 do not effect for X and Y.

```
In[9]:= X^9
Out[9]= X^9
```

We define 0-th power of an algebraic variable to be 1.

```
In[10]:= X^0
Out[10]= K9[1]
```

Since X, Y are indeterminate, you can not substitute a constant to these variables.

```
In[13]:= X = K9[2]
Set::wrsym:Symbol X is Protected
```

6. Mixed operations with integers or constants of base field.

Let GF9 be a Galois field of order 9 with base field GF3. Regular representations of GF3 , 0, 1, 2 are correspond to those of GF9. So

```
In[3]:= GF3[2] + GF9[7]
Out[3]= GF9[6]
```

and

```
In[4]:= GF9[2] + GF9[7]
Out[4]= GF9[6]
```

have same results. If you consider that GF9 is the 3-dimensional vector space over GF3, then addition of vectors and scaler product work as a vector space without any modification.

The Galois Field Package allows addition and multiplication with integers. A integer k in a mixed expression is recognized as $\sum_{1 \le i \le k} 1$, where 1 is the multiplicative identity of the field. Hence k is translated to the regular representation k (mod p), where p is the characteristic of the field.

```
In[4]:= 3 + GF9[4]
Out[4]= GF9[4]
```

```
In[5]:= 4 GF9[3]
Out[5]= GF9[3]
```

```
In[6]:= DeclareVariables[GF9,x]
In[7]:= 5 x
Out[7]= x GF9[2]
```

This feature guarantees that the theorem 7 works for both variables and algebraic variables.

```
In[11]:= Expand[ (x+y)^3]        (x, y are variables of GF9)
Out[11]= x^3 + y^3
In[12]:= Expand[ (X + Y)^3]           (X, Y are algebraic variables of GF9)
Out[12]= X^3 + Y^3
In[13]:= Expand[ (u + v)^3 ]    ( u, v are just symbols in Mathematica)
Out[13]= u^3 +3u^2v + 3uv^2 +v^3
```

and also Derivation over a finite field works

```
In[8]:= DeclareVariables[GF9,a,b,c]
In[9]:= DeclareAlgebraicVariables[GF9,X]
In[10]:= D[a X^2 + b X + c, X]
Out[10]= b + a X GF9[2]
```

7. Making Galois field objects

When you declare a Galois field of type II or type III, there must exist the Galois field object for the field. Once you make a object, you can just declare the Galois field with the algebra name.
A type II Galois field object is made by the following function:
For a type II field

MakeGaloisField[*al*, *ord*, *exd*, *ipp*]

al: algebra name

ord: order of base field

exd : extention degree

ipp: a coefficent list of a primitive irrducible polynomial

```
In[1]:= MakeGaloisField[K16,2,4,{1,1,0,0,1}]
   Galois field object K16 was saved.
```

This function makes an Galois field object with algebra name K16 = GF(2^4) over the base field GF(2) using a primitive irreducible polynomial $1+x+x^4$.
If we try to make an object which aleady exists, then we will have the following message:

```
     Galois field object K16 already exists.
     K16 can not be made.
```

A type III Galois field object is made by the same function but the second argument is differnt.

For a type III field

MakeGaloisField[*al*, *bal*, *exd*, *ipp*]

al: algebra name

bal: algebra name of a base field:

exd: extention degree

ipp: a coefficent list of a primitive irrducible polynomial

The second argument of Type III MakeGaloisField must be an algebra name of base field. The base field must have been declared when you make. If the base field has not been declared, you will have the following message:

```
In[3]:= MakeGaloisField[bb、base,2,{2,2,1}]
 base has not been declared.
 bb can not be made.
```

Note that if the angebra name which you want to make is already declared, you can not make Galois field object as the name.

```
In[4]:= MakeGaloisField[bb,base,2,{2,2,1}]
  bb has been declared already.
  bb can not be made.
```

So, if the conditions

1) the base field is declared,
2) algebra name of object (the first argument) is not declared

are satisfied , then you can make a Galois field object.

```
In[5]:= MakeGaloisField[bb,base,2,{2,2,1}]
    Galois field object bb was saved.
```

8. Declaration of Galois fields

When we finish to set up environments to use a Galois field package mentioned in the section 2, we next have to declare a Galois field using the function DeclareGaloisField with algebra name and some parameters of the field.

For a prime field , it is easy to declare

For a prime field

```
DeclareGaloisField[ algebra name,  order ]
```

The order must be a prime number p for declaring a prime field. This declares a type I Galois field and calculations of constants are always done under the modulus p.

```
In[1]:= DeclareGaloisField[Z5,5]
  Z5 was declared.
    Order is 5
    Characteristics is 5

In[2]:= Z5[3]^3 + Z5[4]^2 + Z5[2]^(-2)
Out[2]= Z5[3]

In[3]:= Sqrt[Z5[4]]
Out[3]= Z5[2]
```

We can also declare an algebra with a order of non prime number using this function, then we can have a modular ring Z/(m). The addition and the multiplication in a modular ring work correctly but inverses are not guaranteed.

```
In[5]:= DeclareGaloisField[Z6,6]
In[6]:= Z6[2] + Z6[3]^3
Out[6]= Z6[5]

In[6]:= Z6[2]^-1
Out[6]= Z6[4]
```

In Z/(6), the inverse of 2 does not exist.   We should not compute   inverses in the algebra.

For an extension field (type II or III)

```
DeclareGaloisField[ algebra name ]
```

This is for declaring type II and III Galois fields.   If there exist the Galois field object of the algebra name, then this function loads a file of the object.   If there does not exist the object then the Galois field Package give the following message:

```
In[1]:= DeclareGaloisField[ abc ]
  Galois field object 'abc' does not exist.
```

When the object exists you will have the following message:

```
In[2]:= DeclareGaloisField[GF16]
  GF16 was declared.
    Order is 16
    Characteristic is 2
    Irreducible Polynomial is {1, 1, 0, 0, 1}
    Base Field is GF2
    Extension Degree is 4
  GF2 was declared.
    Order is 2
    Characteristics is 2
```

Then it is ready to compute over GF(16), for example,

```
In[3]:= GF16[13]^4 + GF16[14] + GF16[6]
Out[3]= GF16[9]

In[4]:= Sqrt[GF16[12]]
Out[4]= GF16[8]
```

For an extension field (type IV)

DeclareGaloisField[*al, bal, exd, ipp* **]**

al : *algebra name*

bal : *algebra name of a base field:*

exp : *extention degree*

ipp : *a coefficent list of a irrducible polynomial*

This declaration gives an extension field without Galois field object. However, we need a base field to be declared before execute this function. This type of Galois field does not have any table to calculate, so this is not recommended for small Galois filed.
Note: the irreducible polynomial is not necessary to be primitive. Furthermore if we put a reducible polynomial then we will have a polynomial ring.

```
In[15]:=  FX = CoefficientList[ 1+ x^7 + x^127, x]
In[6]:= DeclareGaloisField[FF, GF2,517, FX ]
GF2 has been already declared.
FF was declared.
   Order is 170141183460469231731687303715884105728
   Characteristic is 2
   Irreducible Polynomial is
  {1, 0, 0, 0, 0, 0, 0, 1, 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, 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, 1}
   Base Field is GF2
   Extension Degree is 127

In[7]:= FF[132] + FF[321]
Out[7]= FF[453]
```

Show declared algebra names

```
DeclareGaloisField[]
```

The function DeclareGaloisField without arguments gives the list of algebra names which are currently declared.

```
 In[8]:= DeclareGaloisField[]
Out[8]= { FF, GF2, K4 }
```

```
ClearAlgebra[al1, al2, al3,...]
     al1, al2, al3 : algebra names
```

When we declare an finite field, Galois field package defines some symbols and functions in Global context. For example,

```
In[2]:=DeclareGaloisField[GF2,2]
GF2 was declared.
  Order is 2
  Characteristic is 2
In[3]:= ?Global`*
AdditionGF2  ConGF2Q  GF2   NZVGF2Q    Quit       VGF2Q
AlVGF2Q   Exit   MultipleGF2  PowerGF2
```

If you want to remove those, You can do without using Remove function for each.

```
In[4]:= ClearAlgebra[GF2]
In[5]:= ?Global`*
Exit Quit
```

If you do not want to remove variables which are declared by DeclareVariables or DeclareAlgebraicVariables, just use the option `RemoveVariables -> False` .

```
In[6]:= ClearAlgebra[GF2, RemoveVariables->False]
```

But the attributes of the variables (algebra names) are erased.

9. Attributes of Galois field

Declared Galois field has following six attributes including algebra name:

algebra name
characteristic
order
extension degree
irreducible polynomial
base field

The last three of them are not defined for prime fields.

Characteristic[*algebra name*]

If we declare a Galois field as follow:

```
In[16]:= DeclareGaloisField[GF16]
GF16 was declared.
   Order is 16
   Characteristic is 2
   Irreducible Polynomial is {1, 1, 0, 0, 1}
   Base Field is GF2
   Extension Degree is 4
GF2 was declared.
   Order is 2
   Characteristic is 2
```

Then we have

```
In[17]:= Characteristic[GF16]
Out[17]= 2
```

Order[*algebra name*]

gives the order of the Galois field.

```
In[18]:= Order[GF16]
```

```
Out[18]= 16
```

```
ExtensionDegree[ algebra name ]
```

returns the extension degree over the base field.

```
In[19]:=  ExtensionDegree[GF16]
Out[19]= 4
```

```
BaseField[ algebra name ]
```

returns the algebra name of the base field.

```
In[20]:= BaseField[GF16]
Out[20]= GF2
```

10. Variables on Galois fields

There are two kinds of variables in The Galois field package. The first is a variables which represent an element of a Galois field, the second is a symbol of indeterminate. The first is declared by the function DeclareVariables and the second is declared by DeclareAlgebraicVariables.

```
DeclareVariables[algebra name, Var1, Var2,... ]

DeclareVariables[algebra name]

DeclareVariables[]
```

If a Galois field of the algebra name K is declared, DeclareVariables[K, Var1, Var2] defines variables Var1 , Var2 as symbolic elements on the Galois field K.

```
In[1]:= DeclareVariables[Z5,x,y]
```

Now, x and y are variables on Z5, then we have the following results:

```
In[2]:= 3 x^4 * 4 x^3
Out[2]= x3  Z5[2]

In[3]:= Expand[(x + y)^5]
Out[3]= x + y

In[4]:= Solve[x^2 + Z5[1]== 0 ,x]
Out[4]= {{ x -> Z5[2]  }, { x ->  Z5[3] }}
```

DeclareVariables[*algebra name*] returns the list of the variables which are currently declared with respect to the algebra name.
DeclareVariables[ ] returns the list of all variables declared in this session of the Galois field package.

```
In[20]:= DeclareVariables[K5]
Out[20]= { x, y }

In[21]:= DeclareVariables[K9]
Out[21]= { z, w }

In[22]:= DeclareVariables[]
Out[22]= {x, y, z, w }
```

NonZero[*Var1, Var2,* ... **]**

DeclareVariables[*algebra name, Var1, Var2,* ..., **NonZero->False]**

NonZeroQ[*Var* **]**

NonZero[]

When x and y are declared as variables on a Galois field, the function NonZero[x,y] declare x and y to be nonzero variables. NonZero[] returns the list of the all variables which are currently declared as nonzero variables. NonZeroQ[Var] returns True if Var is nonzero variable, otherwise returns False.

```
In[10]:= DeclareVariables[K5, x, y]
```

```
In[11]:= NonZero[ x ]
In[12]:= NonZeroQ[x]
 True
In[13]:= NonZero[]
 { x }
```

The difference between ordinal variables and nonzero variables is shown below :

```
In[14]:= {x , y}^0
Out[14]= { K5[1] y^zero }

In[15]:= {x , y }^4
Out[15]= { K5[1] , y^4}
```

The zero-th power of nonzero variable returns 1, otherwise returns in the form y^{zero}. The (order - 1)-th power of nonzero variable returns 1, otherwise returns as they are.

```
In[10]:= DeclareVariables[K5, x, y]
In[11]:= NonZero[ x ,y ]
```

can be written by the following one function:

`DeclareVariables[K5, x, y, NonZero -> True]`

Default of the NonZero option is False.

DeclareAlgebraicVariables[*algebra name, Var1, Var2,* **...]**

DeclareAlgebraicVariables[*algebra name* **]**

DeclareAlgebraicVariables[]

This function declares algebraic variables (indeterminacies). Galois field of the algebra name must be declared before executing this function.

```
In[6]:= DeclareAlgebraicVariables[K5,a,b]
```

then a and b are algebraic variables over K5=GF(5). Since an algebraic variable does not represent an element of GF(5). So the theorem 5 and 6 do not work on it, i.e. a^5 is not equal to a , a^4 is not equal to 1. However we define 0-th power of an algebraic variable is equal to 1.

```
In[7]:= 3 a^4 * 4 a^3
```

```
Out[7]= a^7  K5[2]

In[8]:= a^0
Out[8]= K5[1]
```

Let K2=GF(2). Let x, y be just variables and c, d algebraic variables on Z2, respectively. Let m, n be just symbols which are not declared in the Galois field package. We can see the difference.

```
In[1]:= DeclareVariables[K2,x,y]
In[2]:= Expand[(x+y)^2]
Out[2]= x + y

In[3]:= DeclareAlgebraicVariables[K2,c,d]
In[4]:= Expand[(c+d)^2]
Out[4]= c^2 + d^2

In[5]:= Expand[(m+n)^2]
Out[5]= m^2 + 2 m n + n2
```

We can usr the function name **DeclareIndeterminates** instead of **DeclareAlgebraicVariables** .
DeclareAlgebraicVariables[*algebra name*] returns he list of algebraic variables declared on the *algebra name*. DeclareAlgebraicVariables[] returns the list of the all algebraic variables declared in this session of Galois field package.

```
In[6]:= DeclareAlgebraicVariables[K5]
Out[6]= { a, b }
In[7]:= DeclareAlgebraicVariables[]
Out[7]= { a, b , c , d }
```

All algebraic variables are protected in Mathematica. Hence, you can not substitute a value to the algebraic variables.

```
In[8]:=  a = K5[1]
Set::wrsym: Symbol a is Protected.
```

ClearVariables[*Var1, Var2,...* **]**

The function ClearVariables removes variables or algebraic variables in the arguments, including functions and attributes defined with respect to the variable names. It is recommended to use this function insted of Mathematica functions Clear or Remove.

AlgebraName[*expression* **]**

This function returns the name of a Galois field on which the expression (may be a variable) is declared.

```
In[9]:= AlgebraName[x]
Out[9]= K2
In[10]:= AlgebraName[ (x + 1)^2 ]
Out[10]= K2
In[11]:= AlgebraName[{ x, y}]
Out[11]= {K2, K5}
```

11. Utilities

ExponentRepresentation[*Galois field constant, symbol* **]**

This function converts from regular representation to exponent representation with the symbol as a primitive element.

```
In[1]:= ExponentRepresentation[GF16[12],x]
```
Out[1]= x^6

Note: This function does not work for the type I and type IV Galois field.

PolynomialRepresentation[*Galois field constant, symbol***]**

converts from *Galois field constant* of regular representation to its polynomial representation on the symbol.

```
In[2]:= PolynomialRepresentation[GF16[12],x]
```

```
Out[2]= x^2 + x^3
```

we can put a vector representation of Galois field constant in the first argument.

```
In[3]:= PolynomialRepresentation[{1,0,1},x]
Out[3]=  1 + x^2
```

`VectorRepresentation[`*Galois field constant* `]`

converts from *Galois field constant* of regular representation to its vector representation.

```
In[4]:= VectorRepresentation[GF16[12]]
Out[4]= {0,0,1,1}
```

`RegularRepresentation[`*algebra name, Galois field constant* `]`

`RegularRepresentation` converts from a Galois field constant (of any representation) to regular form.

```
In[5]:= RegularRepresentation[GF16,x^6]
Out[5]= GF16[12]

In[6]:= RegularRepresentation[GF16,x^3 + x^2]
Out[6]= GF16[12]

In[7]:= RegularRepresentation[GF16,{0,0,1,1}]
Out[7]= GF16[12]
```

`PolynomialGCD[` *polynomial1, polynomial2* `]`

`PolynomialGCD[` *polynomial1, polynomial2, algebraic variable* `]`

`PolynomialGCD` is modified from Mathematica's original function to work on Galois fields. The function returns the greatest common divisor of the two polynomials over a finite field. The result is always a monic polynomial.

```
In[8]:= PolynomialGCD[x^5+GF9[1], x^3+GF9[1]]
Out[8]= x + GF9[1]
```

PolynomialGCD with three arguments is newly made for Galois field package. This PolynomialGCD works for multivariate polynomials.

```
In[15]:= PolynomialGCD[x^5+GF9[3] y ,x^3+y , x]
Out[15]= y GF9[3] + y^3  GF9[4]
```

PolynomialExtendedGCD[*polynomial1, polynomial2* **]**

`PolynomialExtendedGCD` returns the greatest common divisor d(x) of two polynomials in the arguments $f_1(x)$ and $f_2(x)$, and also returns a(x) and b(x) such that

$$a(x)\ f_1(x)\ +\ b(x)\ f_2(x)\ =\ d(x).$$

The output forms { d(x) , { a(x), b(x) } }. The arguments *polynomial1* and *polynomial2* must be univariate polynomials and the variable of the polynomials must be an algebraic variable.

```
In[10]:= DeclareAlgebraicVariables[GF9,x]
In[11]:= PolynomialExtendedGCD[x^3-GF9[1], x^2-GF9[1]]
Out[11]= {x + GF9[2], {1, x GF9[2]}}
In[12]:= Expand[(x^3-GF9[1]) + ( x*GF9[2]) (x^2-GF9[1])]
Out[12]= x + GF9[2]
```

ShiftRegister[*list1, list2***]**

ShiftRegister[*list1, list2, integer***]**

This process is called a feedback shift register. Let *list1* = $\{a_1, a_2, \ldots, a_n\}$ and *list2* = $\{b_1, b_2, \ldots, b_n\}$, The output is $\{0, a_1, \ldots, a_{n-1}\} + a_n * \{b_1, b_2, \ldots, b_n\}$. `ShiftRegister[list1,list2,k]` executes `ShiftRegister[list1,list2]` k times.

```
In[5]:= ShiftRegister[GF2[{1,0,1}], GF2[{0,1,1}]]
Out[5]= {0, GF2[0], GF2[1]}
```

PthRoot[*Galois field constant* **]**

returns the p-th root of the Galois field constant (regular represent) , where p is the characteristic of the field. Note: p-th root of an element x in a prime field is x.

```
In[10]:= PthRoot[ Z5[3] ]
```

```
Out[10]= Z5[3]

In[11]:= PthRoot[ GF9[5] ]
Out[11]= GF9[6]
```

Note: This function does not work for the type IV Galois field.

Variables[*expression***]**

Variables[*algebra name, expression* **]**

Mathematica's original function Variables returns all symbols, including Galois field constants. Therefor the function Variables is modified in the Galois filed package. This function in Galois field package does not returns Galois field constants.

```
In[12]:= Variables[ X^2 + Z5[2] X Y + Y^2 ]
Out[12]= {X, Y}
```

If we use `Variables[`*algebra name, expression*`],` it executes faster than the function `Variables[` *expression* `]` without algebra name.

```
In[13]:= Variables[Z5, X^2 + Z5[2] X Y + Y^2 ]
Out[13]= {X,Y}
```

Polynomialize[*list* , *variable***]**

This function is the inverse process of the function `CoefficientList.`

```
In[14]:= Polynomialize[{Z5[1], 0 , Z5[2], 1}, x]
```
Out[14]= Z5[1] + Z5[2] x^2 + x^3

Sqrt function is also modified for working on the Galois fields.

12. Factor

`Factor` and `FactorList` functions are also modified in the Galois field package to work on finite fields. Since these functions are in a sub-package GaloisFieldFactor.m, in order to use `Factor` and `FactorList` over a finite field, we have to load the package `GaloisFieldFactor.m.`

```
In[20]:= << GaloisFieldFactor.m
GaloisFieldFactor.m was loaded
```

Then we can factor a polynomial over a finite field.

```
In[21]:= DeclareAlgebraicVariables[GF9,Y]
In[22]:= Factor[Y^2 +Y^5 +Y GF9[7] +Y^4 GF9[7] +GF9[8] +Y^3 GF9[8]]
Out[22]= (Y + GF9[1])^3 (Y + GF9[5])^2
```

The variable of a polynomial must be an algebraic variable, never be a variable.

```
In[21]:= FactorList[Y^2 + Y^5+Y GF9[7] + Y^4 GF9[7] + GF9[8]+Y^3 GF9[8]]
Out[21]= {{Y + GF9[1], 3}, {Y + GF9[5], 2}}
```

13. On Development of Applications

When we make an application program using the Galois field package, there are some thing to note. Let us consider an example. We have the following deffinition and theorem on finite fields:

Definition. *If an element a of GF(q) satisfiels* $a^i \neq 1$ *for all* $i = 1,2,..., q\text{-}2$, *then a is called a primitive element of GF(q).*

Theorem 8. *An non-zero element a of GF(q) is a primitive element if and only if* $a^{d_i} \neq 1$ *for all factors* $d_1, d_2, ...d_r$ *of q-1.*

Using Theorem 8, We show a function to test a constant of a Galois field to be a primitive element. PrimitiveQ is designed for returning True if the argument is a primitive element, otherwise returning False.

```
(1) PrimitiveQ[x_]:=
(2)   Block[ {q,d,i},
(3)    q = Order[AlgebraName[x]];
(4)    d = FactorInteger[q - 1];
(5)    For[ i= 1, i<= Length[d], i++,
(6)      If[ x^((q-1)/d[[i,1]]) == 1,
(7)         Return[False];
(8)      ,];
(9)    ];
(10)   Return[True];
(11) ];
(12) PrimitiveQ[x_]:= False /; x == 0;
```

Look at the lines (6) and (12). There, the left hand side of == is a symbol for a Galois field constant but the right hand side is just 1. Only for 0 and 1 , == (Equal function) is modified to test Galois field constant with them. So we can write the following logical equation:

```
                  x == 1 ,  x != 1 ,  x == 0 ,  x != 0
```

However, if the left hand side contains Galois field variables or algebraic variables, then the above equations do not work. In such case, we have to use === (`Same` function) like :

```
    If[  Expr === 1 || Expr === AlgebraName[Expr][1].
```

When we declare an algebra, foe example GF2, then the following symbols are defined in Global context:

```
In[3]:= ?Global`*
AdditionGF2 ConGF2Q   GF2   NZVGF2Q    Quit       VGF2Q
AlVGF2Q    Exit      MultipleGF2      PowerGF2
```

In the symbols,

```
AdditionGF2 ConGF2Q    GF2        NZVGF2Q    VGF2Q
AlVGF2Q    MultipleGF2 PowerGF2
```

are used by Galois field package. These symbols are protected and not cleared unless executing `ClearAlgebra[GF2]`.

Next, we consider a program to obtain the order of a finite field element, that is, to return minimum integer k such that $a^k = 1$, for a Galois filed constant a.

```
(1) Unprotect[Order];
(2) Order[alname_[x_Integer]]:=
(3)   Block[{},
(4)      Cycle[ele_,k_Integer]:=  Block[ {d,i},
(5)         d = FactorInteger[k];
(6)         If[k == d[[1,1]], Return[k],];
(7)         For[ i= 1, i<= Length[d], i++,
(8)            If[ele^(k/d[[i,1]]) == 1,
(9)                       Return[Cycle[ele,kd]],  ];
(10)         ];
(11)        Return[k];
(12)    ];
(13)    Return[Cycle[alname[x],Order[alname]-1]];
(14)  ] ;
(15) Order[alname_[0]] := Return["Not available"];
(16) Order[alname_[1]] := Return[1];
(17) Protect[Order];
```

The function name "Order" is already used twice, one is in original Mathematica and the other is in the Galois field package (order of a field). Since all function names of Mathematica including functions in the Galois field package are protected, we can not use the symbol "Order' at the left hand side of := (Set function). Therefore we have to release the protection to use such symbol, see (1). And we have to write the argument explicitly which distinguish from other Order functions, see (2). Note that the Order function in (13) is the function to obtain the order of a field. This kind of program is a little dangerous, we may break other Order function's operation. It is not recommended for Mathematica beginners.

Ryoh Fuji-Hara
Graduate School of System and Information Engineering , University of Tsukuba
Tsukuba, Ibaraki, Japan 305
Phone: +81-298-53-5088 Fax: +81-298-53-5070
E-mail: fujihara@shako.sk.tsukuba.ac.jp

First edition , August 10, 1993
Second Edition , June 2008