2016-01-20-part2-finite-fields

William A. Stein

1/20/2016

1 Finite Fields

1.1 William Stein

1.2 Jan 20, 2016 (part 2)

A finite field is a field that is finite. The most basic slightly nontrivial mathematical facts about finite fields are:

- Theorem: There exists a unique, up to isomorphism, finite field $_{p^n}$ of each prime power order.
- Theorem: The automorphism group of $_{p^n}$ is cyclic of order n, generated by the Frobenius map $x \mapsto x^p$.

Finite fields are incredibly useful, and frequently arise natural in algebraic number thoery as quotient rings of the form $\mathcal{O}_K/\mathfrak{p}$, where $\mathfrak{p}$ is a nonzero prime ideal in the ring of integers of a number field.

Sage is extremely powerful at working with finite fields (for state-of-the-art research). Sage builds on several very fast/powerful C/C++ libraries for finite field arithmetic, and polynomial arithmetic over finite fields, including Givaro, Singular, FLINT, Pari, and NTL. Different libraries are used for different field sizes.

In practice, some computational issues that arise when you work with finite fields are mainly:

- Choosing a "nice" or "canonical" polynomial $f(x)$ to define the finite field, $_{p^n} \approx_p [x]/(f)$. There are many competing issues here. E.g., cryptographers care desparately about arithmetic speed, especially when $p = 2$; others care about different people agreeing easily on a common choice efficiently (I remember Lenstra et al. getting deeply obsessed with this problem a few years ago). Still others, care about
- Working with the lattice of extensions of $_p$. If you end up with elements in a choice of $a \in_{p^2}$ and $b \in_{p^3}$, and and then suddenly want to work with $a + b$ what to do? This can get very subtle and interesting. I first encountered using such a thing in Magma and was very impressed. For example, there is a frustrating paper about how Magma magically does this, in which they talk about how awesome their approach is, but seem to actually not tell you what it is (and Magma is closed source). I think David Roe and others figured out something just as good that is in Sage.

- Performance: Make computing $a+b$ and $a \cdot b$ in $_{p^n}$ very, very fast. This is extremely interesting, with a wide range of techniques depending on p and n. For example, if p^n is really small, just make a lookup table (that is in cache).
- Arithemetic over finite fields, e.g., with polynomials. This is huge: linear algebra, all polynomial arithmetic, Groebner basis, etc., all over finite fields. Implementations (in Sage) can easily have little to do with how basic arithmetic is implemented (e.g., large degree polynomial multiplication might be done via a Fourier transform, and matrix multiplication may be done by breaking matrices up into blocks and multiplying them using floating point numbers!).

ASIDE: Sage-devel post that appeared right when I write this complaining about the requirment when creating finite fieldsthat you explicitly name the generator

1.3 Making finite fields directly

The Sage documentation explains how you can make finite fields either by directly defining them, or by constructing them as a quotient of the ring of integers of a number field by a prime ideal. I'll show you how to do both below.

1.3.1 The basics

Write GF(p^n,' a')*orFiniteField*(p^n,' a')*tomakethe*(*uniqueuptoisomorphism*)*finitefieldofthatsize*.

```
R.<x> = GF(3)[]
GF(x^2 + 2, 'a')
```

```
Error in lines 2-2
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘sage/structure/factory.pyx’’, line 364, in
sage.structure.factory.UniqueFactory.__call__
(/projects/sage/sage-6.10/src/build/cythonized/sage/structure/factory.c:1245)
    key, kwds = self.create_key_and_extra_args(*args, **kwds)
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/sage/rings/finite_rings/constructor.py’’, line 428, in create_key_and_extra_args
    order = Integer(order)
  File ‘‘sage/rings/integer.pyx’’, line 662, in sage.rings.integer.Integer.__init__
(/projects/sage/sage-6.10/src/build/cythonized/sage/rings/integer.c:5805)
    set_from_Integer(self, otmp(the_integer_ring))
  File ‘‘sage/rings/polynomial/polynomial_element.pyx’’, line 1091, in
sage.rings.polynomial.polynomial_element.Polynomial._integer_ (/projects/sage/sage-6.10/sr
c/build/cythonized/sage/rings/polynomial/polynomial_element.c:11994)
    raise TypeError(‘‘cannot coerce nonconstant polynomial’’)
TypeError: cannot coerce nonconstant polynomial
```

```
GF(3^2, modulus=x^2+1, names='a')
```
```
Finite Field in a of size 3^2
```

1.3.2 A Prime finite field (so $n = 1$)

```
GF(3)
```
```
Finite Field of size 3
```

```
F = GF(3)
list(F)
```
```
[0, 1, 2]
```

```
type(F)
```
```
<class
'sage.rings.finite_rings.finite_field_prime_modn.FiniteField_prime_modn_with_category'>
```

```
2*2
```
```
4
```

```
F(2) * F(2)
```
```
1
```

Arithmetic in small finite fields is reasonably fast.

```
a = F(2)
%timeit a*a
```
```
625 loops, best of 3: 99.2 ns per loop
```

```
%timeit a+a
```
```
625 loops, best of 3: 109 ns per loop
```

Exercise: How many additions/multiplications with elements of GF(3) can you do in one second?
Memorize the answer to this question it will help you a lot when thinking about how long things should take.

```
# figure it out now here --
```

For comparison try Sage number field elements:

```
K.<a> = NumberField(x^2 - 2)
%timeit a*a
%timeit a+a
```
```
625 loops, best of 3: 969 ns per loop
625 loops, best of 3: 832 ns per loop
```

Quick Exercise: Easy how many number field operations per second?

Also one with big p.

```
GF(next_prime(10^100))
```

```
Finite Field of size 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000267
```

1.3.3 Non-prime finite fields, so $n > 1$

```
# Expected to get error, as Nathann complains about above...
GF(9)
```

```
Error in lines 2-2
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘sage/structure/factory.pyx’’, line 364, in sage.structure.factory.UniqueFactory.__call__ (/projects/sage/sage-6.10/src/build/cythonized/sage/structure/factory.c:1245)
    key, kwds = self.create_key_and_extra_args(*args, **kwds)
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-packages/sage/rings/finite_rings/constructor.py’’, line 479, in create_key_and_extra_args
    raise ValueError(‘‘parameter 'conway' is required if no name given’’)
ValueError: parameter 'conway' is required if no name given
```

```
F.<a> = GF(9)  # create the finite field -- just like making a \
   number field
print F
```

```
Finite Field in a of size 3^2
```

```
list(F)
```

```
[0, a, a + 1, 2*a + 1, 2, 2*a, 2*a + 2, a + 2, 1]
```

```
type(F)  # uses Givaro, which probably uses a lookup table ot make \
   this fast for small cardinality
```

```
<class 'sage.rings.finite_rings.finite_field_givaro.FiniteField_givaro_with_category'>
```

```
(2+a)^5
```

```
2*a + 1
```

```
(2+a)^3
```

```
2*a
```

```
((2+a)^3)^3
```
```
a + 2
```

```
F.modulus()
```
```
x^2 + 2*x + 2
```

```
a.minimal_polynomial()
```
```
x^2 + 2*x + 2
```

```
a^2 + 2*a + 2
```
```
0
```

Exercise: Is arithmetic in GF(9) in Sage "fast"? How does it compare to GF(3) above?

```
# Bonus: surprisingly, this works :-)
GF(3)[sqrt(2)]
```
```
Finite Field in sqrt2 of size 3^2
```

1.3.4 Lattice of finite fields

```
F2.<a> = GF(3^2)
F3.<b> = GF(3^3)
a + b  # bummer, sage doesn't do this...
```
```
Error in lines 3-3
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘sage/structure/element.pyx’’, line 1651, in
sage.structure.element.RingElement.__add__
(/projects/sage/sage-6.10/src/build/cythonized/sage/structure/element.c:15852)
    return coercion_model.bin_op(left, right, add)
  File ‘‘sage/structure/coerce.pyx’’, line 1069, in
sage.structure.coerce.CoercionModel_cache_maps.bin_op
(/projects/sage/sage-6.10/src/build/cythonized/sage/structure/coerce.c:9736)
    raise TypeError(arith_error_message(x,y,op))
TypeError: unsupported operand parent(s) for '+': 'Finite Field in a of size 3^2' and
'Finite Field in b of size 3^3'
```

```
%magma    /* magma does */
F2<a> := FiniteField(3^2);
F3<b> := FiniteField(3^3);
a + b
```

```
$.1^297
```

```
F2.<a> = GF(3^2)
F3.<b> = GF(3^3)
F6.<c> = GF(3^6)
```

```
# Sadlly, there is no "embeddings" command to produce all finite \
   field morphisms from
# one field into another, which would be EASY to write (see below):
F2.embeddings
```

```
Error in lines 3-3
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘sage/structure/parent.pyx’’, line 855, in sage.structure.parent.Parent.__getattr__
(/projects/sage/sage-6.10/src/build/cythonized/sage/structure/parent.c:8043)
    attr = getattr_from_other_class(self, self._category.parent_class, name)
  File ‘‘sage/structure/misc.pyx’’, line 253, in
sage.structure.misc.getattr_from_other_class
(/projects/sage/sage-6.10/src/build/cythonized/sage/structure/misc.c:1667)
    raise dummy_attribute_error
AttributeError: 'FiniteField_givaro_with_category' object has no attribute 'embeddings'
```

```
v = F2.polynomial().roots(ring=F6); v
```

```
[(2*c^5 + 2*c^3 + c^2 + 2*c + 2, 1), (c^5 + c^3 + 2*c^2 + c + 2, 1)]
```

```
phi = Hom(F2, F6)(v[0][0])
phi
```

```
Ring morphism:
  From: Finite Field in a of size 3^2
  To:   Finite Field in c of size 3^6
  Defn: a |--> 2*c^5 + 2*c^3 + c^2 + 2*c + 2
```

```
phi(a)
```

```
2*c^5 + 2*c^3 + c^2 + 2*c + 2
```

```
psi = Hom(F3, F6)(F3.polynomial().roots(ring=F6)[0][0])
psi
```

```
Ring morphism:
  From: Finite Field in b of size 3^3
  To:   Finite Field in c of size 3^6
  Defn: b |--> 2*c^5 + 2*c^4
```

```
psi(b)
```

```
2*c^5 + 2*c^4
```

```
# Finally, compute the sum in F6:
phi(a) + psi(b)
```

```
c^5 + 2*c^4 + 2*c^3 + c^2 + 2*c + 2
```

However, Sage does support working in $_p$, which is possibly very nice and solves the same problem.

```
Fbar = GF(3).algebraic_closure()
Fbar
```

```
Algebraic closure of Finite Field of size 3
```

```
g2 = Fbar.gen(2); g2
g3 = Fbar.gen(3); g3
g2 + g3
```

```
z2
z3
z6^5 + 2*z6^4 + 2*z6^3 + z6^2 + 2*z6 + 1
```

```
(g2+g3).minpoly()
```

```
x^6 + x^4 + 2*x^3 + x^2 + x + 2
```

Exercise: Whenever you try anything in software you should be very worried that the implementation sucks. You should run some basic benchmarks and compare with what you know. Never trust anything (especially do not use closed source software). Is Fbar slow or fast? Try some simple benchmarks right now.

Next time, finite fields from number fields.