2016-01-22-finite-fields-second-part

William A. Stein

1/22/2016

Contents

1 Finite Fields (more)

1.1 William Stein

1.2 Jan 22, 2016

Motivation: if you plan to ever do any computations over finite fields, today is going to make your life easier later. Simple as that.

1.2.1 Update on the “saga” of GF(9).

```
# Recall
GF(9)   # expected "boom"
```

```
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
```

```
# but...
GF(9,'a')
```

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

Then Nathann Cohen complained, which led to a huge thread and this ticket: http://trac.sagemath.org/ticket/1

- some discussion
- a rude/mean comment from Nathann directed at me
- ticket closed
- another ticket: http://trac.sagemath.org/ticket/17569

It looks like doing GF(9) might in the near future construct the quadratic subfield of $\bar{}_3$. Is this a good idea? What is $\bar{}_p$ in Sage anyways?

Let’s find out!

1.2.2 The lattice of finite fields?

```
F2.<a> = GF(3^2)
F3.<b> = GF(3^3)
a + b  # Sage doesn't magically 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'
```

1.2.3 Bummer?

No, not really

There's no reason to expect a+b to make any sense in Sage, since we have two finite fields above with two different variable. Also, if the names were the same, it also wouldn't make sense, because then *a* would be the generator of $_{3^2}$ and $_{3^3}$ simultaneously.

Incidentally, Magma doesn't care what you call things, and does make sense of adding the *a* and *b* defined above:

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

```
$.1^297
```

1.2.4 Back to Sage

At least we can do everything very explicitly and define embeddings.

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

```
# Sadly, 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 859, in sage.structure.parent.Parent.__getattr__
(/projects/sage/sage-6.10/src/build/cythonized/sage/structure/parent.c:8131)
    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'
```

```
# First get the roots of the defining polynomial of F2 in the field \
   F6.
v = F2.polynomial().roots(ring=F6, multiplicities=False); v
```

```
[2*c^5 + 2*c^3 + c^2 + 2*c + 2, c^5 + c^3 + 2*c^2 + c + 2]
```

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

```
def embeddings(E, F):
    """Compute all embeddings from the field E into the field F."""
    H = Hom(E,F)
    return [H(x) for x in E.polynomial().roots(ring=F, \
  multiplicities=False)]
```

```
psi = embeddings(F3,F6)[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
```

```
# this takes a lot of time though:
%timeit phi(a) + psi(b)
```

```
625 loops, best of 3: 120 µs per loop
```

So at least it's possible to play around with different finite fields.

Though, as you might imagine, this "naive" appraoch can get very confusing and problematic! What if you do some more computations, choosing embeddings, and end up with two different ways to get from $_{3^2}$ to $_{3^{12}}$, say? In a program it could happen very naturally, and lead to subtle bugs.

However, Sage does support working in $_p$, which is possibly very nice and solves our problem in a way that doesn't run into subtle bugs/contradictions if things get more complicated.

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

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

```
# Heh, somebody should implement latex(Fbar) -- it would be so easy\
   ...
show(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
```

```
F6, f = Fbar.subfield(6)
F6
f
```

```
Finite Field in z6 of size 3^6
Ring morphism:
  From: Finite Field in z6 of size 3^6
  To:   Algebraic closure of Finite Field of size 3
  Defn: z6 |--> z6
```

```
f(F6.0)
```

```
z6
```

```
F6.0 + g2 + g3
```

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

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.

```
# 1. Benchmark (using %timeit) adding/multiplying g2 with itself \
   here:
```

```
# 2. Benchmark (using %timeit) adding/multiplying g2 with g3 (which \
   results in an element of F6):
```

```
# 3. CRITICAL: How does the benchmark in 2 compare to what we did \
   above?
```

1.2.5 How does $_p$ in Sage work?

The main idea is to use pseudo-conway polynomials, which are conway polynomials without the lexicographic condition.

Definition: Suppose f_n is the minimal polynomial of a generator a of $^*_{p^n}$. For $m < n$, let a^r be the smallest power of a that generators $_{p^m}$, and let f_m be the minimal polynomial of a^r. Say that f_n and f_m are compatible of f_m divides f_n.

We call f_n a pseudo-Conway polynomial if it is compatible with f_m for all $m < n$. This means that you can very easily understand the subfields of $_{p^n}$ and maps between them explicitly in terms of these polynomials.

Of course finding a pseudo-Conway polynomial isn't trivial.

To make it even harder, you could try to find a Conway polynomial which is the same as above, except you require all the f_n to be lexicographically minimal.

Exercise: Compare field creation time.

```
# 1. Something small using timeit (which might be very misleading?) \
    -- just try changing 3 to something small
%timeit GF(3^6,'a')
%timeit GF(3).algebraic_closure().subfield(6)
```

```
625 loops, best of 3: 277 µs per loop

625 loops, best of 3: 641 µs per loop
```

```
# 2. Something bigger: make p a prime with about 10 digits and the \
    extension of degree 6.
p = next_prime(  # figure it out ...)
%time GF(p^6, 'a')
%time GF(p).algebraic_closure().subfield(6)
```

```
# 3. Something bigger: make p a prime with about 20 digits and the \
    extension of degree 6.
```

Next time (again delayed): finite fields from prime ideals in rings of integers of number fields