CoCoALib - Feature #664

Impl small non-prime finite fields (using logs)

11 Feb 2015 17:39 - John Abbott

Status:	Resolved	**Start date:**	11 Feb 2015
Priority:	Urgent	**Due date:**	
Assignee:	John Abbott	**% Done:**	80%
Category:	New Function	**Estimated time:**	0.00 hour
Target version:	CoCoALib-0.99880	**Spent time:**	36.85 hours

Description

Impl small non-prime finite fields (?using log/exp tables?)

Werner+Mario would like these ASAP; especially with char=2.

Related issues:

Related to CoCoALib - Feature #667: factor: multivariate + finite characteristic	**New**	**02 Mar 2015**
Related to CoCoALib - Design #785: finite fields: global register of fields a...	**New**	**12 Oct 2015**
Related to CoCoA-5 - Support #242: CoCoA-5 Projects for students (e.g. credit...	**In Progress**	**28 Sep 2012**

History

#1 - 13 Feb 2015 20:46 - John Abbott

When char=2 we can readily use (binary) integers to represent field elements.

The idea is to find a generator theta of the cyclic group of units in the finite field; this will also be a field generator. Then the integer 10 = 1010 (binary) represents theta^3+theta^1.

The idea for finding a generator is simply to try all possible monic polynomials of the required degree in increasing lex order until one is found whose root is a generator of the cyclic group (involves factorizing the integer p^deg - 1, not a problem for small values).

#2 - 11 Jun 2015 15:26 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 30

JAA has implemented a first prototype (following the model RingFpImpl and SmallFpImpl). The computational part uses "vectors" of SmallFpImpl::value_t to represent the field element; it appears to work correctly (*i.e.* passes a few tests).

The slightly tricky part for RingFqImpl is that the ring is a quotient ring, presumably of F_q[x]. But where does F_q[x] come from, and what should the symbol for the indet be?

Should there be a common F_q[x] for all simple algebraic extensions of F_q? This would mean that all primitive elements of all algebraic extensions of F_q will have the same symbolic name -- probably pretty awkward for the user. :-(

The pseudo-ctor for RingFqImpl could create F_q[anon] where anon is a new anonymous symbol; this would be pretty ugly in printing.

Another possibility would be to have a mechanism similar to that for anonymous symbols but with a specified head: *e.g.* alg[1], alg[2] and so on. This is OK if the user does not want to use such names himself.

#3 - 12 Jun 2015 12:03 - John Abbott

- % Done changed from 30 to 40

Now I have a (partly?) working CoCoALib ring which wraps my previous code.

#4 - 01 Jul 2015 17:57 - John Abbott

- Target version changed from CoCoALib-0.99536 June 2015 to CoCoALib-0.99540 Feb 2016

#5 - 30 Jul 2015 17:36 - John Abbott

- Priority changed from High to Urgent

#6 - 12 Oct 2015 17:13 - John Abbott

If CoCoALib offers a function which creates a (non-prime?) finite field given just its cardinality (or maybe characteristic and extension degree?) then we can choose freely how that field is represented, and what it is an extension of.

Such a "freely determined" finite field should be a simple algebraic extension of F_q[x] where F_q is the (unique?) prime field of characteristic radical(q); see issue #785.

#7 - 14 Oct 2015 09:03 - Anna Maria Bigatti

If we construct a F_p[x]/(f) would it be possible to convert it internally to your F_q? (I'm very ignorant on this subject)

#8 - 02 Dec 2015 15:08 - John Abbott

Certainly an explicit quotient ring which is a finite field will be isomorphic to one of "my" finite fields. It might even be possible to make the quotient ring recognize when this happens, and then create one of my fields inside itself. To take advantage of this it would also have to create a pair of "complementary" isomorphisms between the two fields; this involves factorizing a univariate polynomial over a (small prime) finite field.

I'd need to implement this to understand how complicated it might become in code.

#9 - 02 Dec 2015 15:13 - John Abbott

I have just tried a simplistic speed test (on the "new" computer here in Kassel). The aim was to see how costly it is to convert a "compressed" value into vector<int> where the compressed value is simply "sum of a[i]*b^i" where the base "b" is at least the characteristic of the field.

It simply converted millions of machine integers into vector<int> being a representation in base 3; I then repeated the loop but used bit shifting (by 2 bits) instead of division... so strictly it produced a representation in base 4 instead of base 3. Anyway, division-and-remainder took about 200s whereas bit-shifting took about 36s; that's almost 6 times faster.

ADDENDUM: with base=11, and 4 bits the times were 104s vs 32s

I'm not yet sure what this implies for making a good/fast implementation using log/exp tables.

#10 - 02 Dec 2015 15:50 - John Abbott

Here are some options for implementing extensions of (prime) finite fields:

- **(A)** use a CoCoALib quotient ring
- **(B)** use a special impl with vector<int> repr for elements

- **(C)** use a special impl with "compressed" repr for elements (vector compressed into long)
- **(D)** use a special "compressed" impl for char 2

(A) already exists; it has no particular limits, but is (very probably) quite slow.
(B) largely exists as a "dirty" prototype; assumes a single generator, only limit is that characteristic must be suitable for SmallFpImpl; should be faster than **(A)**, but I'm not sure how much faster.
(C) is the "most obvious" specialized impl if we want to use log/exp tables for multiplication, but previous comment suggests that conversion between compressed and uncompressed (for addition/substraction) may be costly; there is a limit on cardinality of the extension field;
(D) is the impl Werner desires; the big advantage compared to **(C)** is that addition can be achieved via xor.

The compression/decompression problem of **(C)** could be avoided by using the "log(x+1)" trick which enables one to compute x+y = x*(y/x+1); in other words, compute y/x as log(y)-log(x), use the new table to get log(y/x+1), then add log(x), and finally exp the result (if not using a logarithmic repr).

#11 - 02 Dec 2015 17:17 - John Abbott

A very simple speed test: compute the order of the field/group generator by naive repeated multiplication.
Prototype impl **(B)** is typically 10-20 times faster than impl **(A)**.

Since this naive test involves only multiplication, I expect that a log/exp impl would be much faster.

I should also devise a more realistic test; perhaps naive multiplication of two random univariate polys? That will involve as many additions as multiplications (but no divisions).

Another possible test could be gaussian reduction of a random matrix?

#12 - 03 Dec 2015 10:09 - John Abbott

- % Done changed from 40 to 50

Given the cost of converting between a "compressed" format and a "vector" format, it is probably better to represent values as their logs and use a "Zech table" for addition (as hinted at in the second part of comment 10).

In fact, I am quite tempted to encode the values as follows:

- 0 encodes as 0
- n (non-zero) encodes as 1+log(n) giving a value from 1 to p-1

The advantage of this "peculiar" representation is that 0 and 1 encode as themselves, and furthermore the encoding of these two special values is independent of the finite field (much as happens for the base finite fields).

#13 - 09 Dec 2015 17:02 - John Abbott

JAA will try to implement this before xmas (2015!)

But first some Gluehwein ;-)

#14 - 17 Dec 2015 16:19 - John Abbott

- % Done changed from 50 to 70

JAA now has a prototype impl which still needs lots of cleaning, but some simple tests pass :-)

I'm hoping to clean and document and check in before tomorrow evening.

#15 - 18 Dec 2015 16:26 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 70 to 80

I have checked in the code I currently have. no doc, no examples, no tests.
An unofficial, simple example (Fq4.C on my computer) seems to work OK.

#16 - 27 Jan 2016 16:12 - John Abbott

Aha! Now you have to click on Edit to add a new progress message (previously it was Update, IIRC).

I have added one simple test (but still tough enough to have unearthed a silly (embarassing) bug last night).
I have added one simple example which I hope covers most of the special extra operations one might need to do to a non-prime finite field.

Still no doc: I am not yet happy with the user interface.

#17 - 27 Jan 2016 16:14 - John Abbott

Currently elements of a RingFq print in a non-standard way. I think it is quite easy to read, so am happy to leave it as a temporary solution until a better idea strikes me.

#18 - 27 Jan 2016 21:16 - John Abbott

Printing of elems of a (non-prime) finite field needs to be sorted out.

In fact the way an element of a (non-prime) finite field prints is quite independent of how the canonical "lift" to the element in the polyring prints out.
So it would be possible to print out the element as a polynomial but with a variable name of our choosing rather than the variable name used in the polyring "sitting above" our quotient ring.

However a user might find it strange that an element of FFq which prints out as 12*sqrt2-17 lifts to 12*x-17, say... assuming we had chosen sqrt2 as the print name inside the quotient ring.

I'm not sure whether this is a good idea or a bad one :-/

#19 - 27 Jan 2016 22:27 - Anna Maria Bigatti

John Abbott wrote:

> However a user might find it strange that an element of FFq which prints out as 12*sqrt2-17 lifts to 12*x-17, say... assuming we had chosen sqrt2 as the print name inside the quotient ring.

If we choose a name I think we should use it also for creating the associated polynomial ring....(but I think sqrt2 is not a valid symbol name)
Leaving the freedom could lead to some confusion.

ADDENDUM refer to issue #568 for valid symbol heads; sqrt2 is now allowed.

#20 - 27 Jan 2016 22:57 - John Abbott

I quite understand the concern about possible confusion.

I am also concerned about creating a "shadow" polyring for every non-prime finite field. My (rather ignorant) guess is that these polyrings will hardly ever be used; they would serve only to "look inside" the structure of an algebraic number.

I suppose the KISS philosophy would favour having the same name in both the polyring and the quotient ring (the alg extn). Perhaps we can implement that, but keep an eye out for possible problems related to the creation of many polyrings (each with its own MemPool).

What name(s) should we use if the user does not supply one? My suggestion would be something like alg[3] or even AlgNum[3], though the longer name could impede readability in a large polynomial with many algebraic coefficients.

#21 - 21 Mar 2016 15:16 - John Abbott

- Target version changed from CoCoALib-0.99540 Feb 2016 to CoCoALib-0.99550 spring 2017

#22 - 21 Sep 2016 18:17 - John Abbott

- Target version changed from CoCoALib-0.99550 spring 2017 to CoCoALib-0.99560

#23 - 06 Nov 2017 15:25 - John Abbott

- Target version changed from CoCoALib-0.99560 to CoCoALib-0.99600

#24 - 03 Aug 2018 16:02 - John Abbott

- Target version changed from CoCoALib-0.99600 to CoCoALib-0.99650 November 2019

#25 - 01 Oct 2019 11:58 - John Abbott

- Target version changed from CoCoALib-0.99650 November 2019 to CoCoALib-0.99800

#26 - 05 Oct 2020 13:08 - John Abbott

- Target version changed from CoCoALib-0.99800 to CoCoALib-0.99850

I'd like to have the time to finish this, but that seems very unlikely in the immediate future. :-(
Postponing (yet again).

#27 - 17 Feb 2022 19:33 - John Abbott

- Related to Support #242: CoCoA-5 Projects for students (e.g. crediti F and tesi) added

#28 - 14 Mar 2023 19:49 - John Abbott

- Target version changed from CoCoALib-0.99850 to CoCoALib-0.99880

I'd need a few free days to finish this (and to remember where I'd got to).
Maybe something for the summer holiday?

Postponing (yet again).
HINT we should also look up *Conway polynomials* (standardized irred polys for group gens)