Status Report: Commutative Algebra in SAGE

Martin Albrecht (M.R.Albrecht@rhul.ac.uk)

Cambridge, USA, October 4th, 2007

Outline

Commutative Algebra

In abstract algebra, commutative algebra studies commutative rings, their ideals, and modules over such rings. Both algebraic geometry and algebraic number theory build on commutative algebra. Prominent examples of commutative rings include polynomial rings **(covered here)**, *rings of algebraic integers* **(see William Stein/Robert Bradshaw's talk)**, *including the ordinary integers Z, and p-adic integers* **(see David Roe's talk)**.

`http://en.wikipedia.org/wiki/Commutative_algebra`

Specifically, we will cover multivariate polynomial rings over fields and commutative rings here.

A Little Bit of History

My ToDo list of October 2006:

- way faster multivariate polynomial arithmetic
  - ETuples: Pyrex $\longrightarrow$ C.
  - faster finite field arithmetic (e.g. Givaro) (nearly done)
  - now we are SO much better than this
- fast *sparse* linear algebra – mostly (reduced) row echelon form – over any field. (g0n, LinBox)
- for crypto: Efficient implementation of $\mathbb{F}_2[x_0, \ldots, x_{n-1}]/\langle \text{FieldIdeal} \rangle$ (PolyBoRi)
- a useable but flexible (extensible) F_4 implementation in SAGE
- F_5 would be very cool. (yeah, right)

Outline

(Planed) Inheritance Tree

Rings:

- CommutativeRing — a commutative ring
  - MPolynomialRing_generic — common operations
    - MPolynomialRing_polydict — generic implementation
    - MPolynomialRing_libsingular — polynomials
      - QuotientRing_libsingular — quotient ring
    - MPolynomialRing_cocoa — linking against CoCoALib
  - QuotientRing_generic — common operations
    - PolynomialQuotientRing — generic implementation
    - BooleanPolynomialRing — PolyBoRi

Ideals:

- Ideal_generic — generic for any ring
  - MPolynomialIdeal — over multivariate polynomials
  - BooleanPolynomialIdeal — PolyBoRi

MPolynomial_generic

sage.rings.polynomial.multi_polynomial_element

- straight forward Python/Cython implementation
- polynomials represented as dictionaries of exponent vectors and coefficients.
- Consider for example the ring $\mathbb{Q}[x, y, z]$ and $f = 5 * x^2y^3 + z^4 - 2$. This boils down to {{0:2,1:3}:5, {2:4}:1, {}:-2}
- basics work over any field/ring
- currently provides e.g. $\mathbb{Z}$.
- very slow

MPolynomial_libsingular

sage.rings.polynomial.multi_polynomial_libsingular

SINGULAR http://www.singular.uni-kl.de/

- implementation linking against SINGULAR directly on C level
- needed to convert SINGULAR to a shared library (changes accepted upstream)
- provided base fields: $\mathbb{F}(p^n)$, $\mathbb{Q}$.
- fast basic arithmetic, fastest I am aware of over $\mathbb{F}(p)$.

```
sage: P.<x,y,z> = PolynomialRing(GF(32003),3)
sage: p = (x + y + z + 1)^20; q = p + 1 # the Fateman fastmult benchmark
sage: r = p*q
Core2Duo 2.33Ghz, Linux; SAGE/Singular: 0.24s, MAGMA: 0.52s
```

- also provided via libSINGULAR: matrices over those polynomial rings, some ideal operations.
- if there is stuff in SINGULAR that needs to be in SAGE directly please speak up!

libSINGULAR isn't as scary as many believe

sage.rings.polynomial.multi_polynomial_libsingular

```
cdef ModuleElement _add_c_impl( left , ModuleElement right ):
    """
    Add left and right.

    EXAMPLE:
        sage: P.<x,y,z>=MPolynomialRing(QQ,3)
        sage: 3/2*x + 1/2*y + 1
        3/2*x + 1/2*y + 1
    """
    cdef MPolynomial_libsingular res

    cdef poly *_l, *_r, *_p
    cdef ring *_ring

    _ring = (<MPolynomialRing_libsingular>left._parent)._ring

    if(_ring != currRing): rChangeCurrRing(_ring)

    _l = p_Copy(left._poly, _ring)
    _r = p_Copy((<MPolynomial_libsingular>right)._poly, _ring)

    _p= p_Add_q(_l, _r, _ring)

    p_Normalize(_p,_ring)

    return co.new_MP((<MPolynomialRing_libsingular>left._parent),_p)
```

Incidently: SINGULAR's $\mathbb{Q}$ implementation

```
sage: n = 3000000
sage: p = ZZ(randint(0,2^n))/ZZ(randint(0,2^n))
sage: q = ZZ(randint(0,2^n))/ZZ(randint(0,2^n))

sage: %time _ = p+q
CPU times: user 2.15 s, sys: 0.00 s, total: 2.15 s
Wall time: 2.16

sage: %time _ = p*q
CPU times: user 3.72 s, sys: 0.00 s, total: 3.72 s
Wall time: 3.78

sage: P.<x,y> = PolynomialRing(QQ,2)
sage: p = P(p); q = P(q)

sage: %time _ = p+q
CPU times: user 4.68 s, sys: 0.01 s, total: 4.69 s
Wall time: 4.69

sage: %time _ = p*q
CPU times: user 0.25 s, sys: 0.00 s, total: 0.25 s
Wall time: 0.25
```

MPolynomial_cocoalib

CoCoALib is GPL'd now!

http://cocoa.dima.unige.it/cocoalib/

- uses [Ap]CoCoALib as backend, see Michael Abshoff's talk
- initial wrapper was written in April 2007 by Michael Abshoff and me withing five hours
- never made it upstream
- will probably provide at least $\mathbb{R}$ and $\mathbb{C}$ as base fields.
- basic arithmetic seems slower than libSINGULAR for $\mathbb{F}_p$ (maybe due to calling overhead)
- many interesting things on todo list but much stuff not quite there yet.

BooleanPolynomial

PolyBoRi http://www.itwm.fraunhofer.de/en/as__asprojects__PolyBoRi/PolyBoRi/

- Uses PolyBoRi by Alexander Dreyer and Michael Brickenstein as backend.
- PolyBoRi is a library for computations in $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$
- applications: crypto, coding theory, ...
- representation of polynomials as zero suppressed decision diagrams (ZDDs): very efficient for large polynomials
- Burcin Erocal is working on the integration, prototype available but cannot distribute yet, because PolyBoRi author's didn't officially release yet (but will soon).

Outline

Basic Arithmetic

It does matter!

- claim: basic arithmetic is way less important than e.g. Gröbner basis calculations. E.g. noone want so multiply large multivariate polynomials. this is not true
- SAGE should be a system to protype algorithms and thus basic arithmetic matters
- SAGE is very fast for multivariate polynomials over finite fields, pretty good for $\mathbb{Q}$ and bad for the rest
- also, SAGE covers basic arithmetic to e.g. prototype Gröbner basis algorithms.
- See **sage.rings.polynomial.toy_buchberger**

Buchberger's Algorithm

sage.rings.polynomial.toy_buchberger

Remember, that G is a Gröbner basis, if $LM(\langle G \rangle) == \langle LM(G) \rangle$. So, we try to find elements in $\langle G \rangle$ (and thus in $LM(\langle G \rangle)$) which are not in $\langle LM(G) \rangle$.

```
LM = lambda f: f.lm()
LT = lambda f: f.lt()
spol = lambda f,g: LCM(LM(f),LM(g)) // LT(f) * f - LCM(LM(f),LM(g)) // LT(g) * g

def buchberger(F):

    G = set(F)
    B = set(filter(lambda (x,y): x!=y, [(g1,g2) for g1 in G for g2 in G]))

    while B!=set():
        g1,g2 = select(B)
        B.remove( (g1,g2) )

        h = spol(g1,g2).reduce(G)
        if h != 0:
            B = B.union( [(g,h) for g in G] )
            G.add( h )

    return Sequence(G)
```

Gröbner Bases

That was nice, however:

Katsura-7 over $\mathbb{Q}$

```
sage: P = PolynomialRing(QQ,7,'x')
sage: I = sage.rings.ideal.Katsura(P)
sage: time gb = I.groebner_basis('toy:buchberger2') # improved version
CPU times: user 256.36 s, sys: 0.36 s, total: 256.72 s
Wall time: 258.92

sage: I = sage.rings.ideal.Katsura(P)
sage: time gb = I.groebner_basis('libsingular:slimgb')
CPU times: user 0.46 s, sys: 0.00 s, total: 0.46 s
Wall time: 0.46
```

Katsura-8 over $\mathbb{Q}$

```
sage: P = PolynomialRing(QQ,8,'x')
sage: I = sage.rings.ideal.Katsura(P)
sage: time gb = I.groebner_basis('libsingular:slimgb')
CPU times: user 5.55 s, sys: 0.00 s, total: 5.56 s
Wall time: 5.57

sage: I = sage.rings.ideal.Katsura(P)
sage: time gb = I.groebner_basis('magma:GroebnerBasis')
CPU times: user 1.00 s, sys: 0.03 s, total: 1.03 s
Wall time: 1.55
```

PolyBoRi/BooleanPolynomialRing

claimed speed from the paper w.r.t to *lex*

Example	var.	eq.	PolyBoRi		Singular	
ctc-5-3	190	354	3.04 s	49MB	32s	69MB
ctc-8-3	298	561	4.8 s	52MB	117s	154MB
ctc-15-3	550	1044	8.04 s	69MB	748s	379MB
aes-10-1-1-4pp	170	184	0.14 s		0.25 s	
aes-7-1-2-4pp	210	255	3.24 s	50MB	18 s	
aes-10-1-2-4pp	288	318	6.7 s	51 MB	1080 s	694 MB

Example	var.	eq.	Magma		Maple
ctc-5-3	190	354	83 s	64 MB	$>$ 1800 s
ctc-8-3	298	561	817s	335MB	
ctc-15-3	550	1044	$>$ 3000 s	$>$ 570 MB	
aes-10-1-1-4pp	170	184	0.92 s	9.25 MB	$>$ 1000 s
aes-7-1-2-4pp	210	255	366s	211 MB	
aes-10-1-2-4pp	288	318	978 s	477 MB	$>$ 70 h

Local Orderings

$x < 1$

If we use the *negative degrevlex* ordering for example, we are computing in the localisation of the polynomial ring at the origin and this allows us to compute multiplicities of zeroes there:

```
sage: A.<x,y> = PolynomialRing(QQ,2,order='degrevlex')
sage: I = Ideal([x^10 + x^9*y^2, y^8 - x^2*y^7])
sage: J = Ideal(I.groebner_basis())
sage: J._singular_().vdim()
83
sage: Ideal(J.radical().groebner_basis())._singular_().vdim()
4
```

This computations tell us that these equations have 83 solutions but only 4 distinct ones. Since the origin is a common solution we may compute the multiplicity by passing to the localisation:

```
sage: B.<x,y> = PolynomialRing(QQ,2,order='negdegrevlex')
sage: I0 = Ideal([B(f) for f in I.gens()])
sage: J0 = Ideal(I0.groebner_basis())
sage: J0._singular_().vdim()
80
```

This means that the origin has multiplicity 80 and the other three roots are simple (multiplicity one).

Block/Product Orderings

Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$ be two ordered sets of variables, $<_1$ a monomial ordering on $\mathbb{K}[x]$ and $<_2$ a monomial ordering on $\mathbb{K}[y]$. The product ordering (or block ordering) $< := (<_1, <_2)$ on $\mathbb{K}[x, y]$ is the following:
$x^a y^b < x^A y^B \Leftrightarrow x^a <_1 x^A$ or $(x^a = x^A$ and $y^b <_2 y^B)$.

```
sage: T = TermOrder('lex', 3)
sage: T += TermOrder('degrevlex', 2)
sage: P.<a,b,c,d,e> = PolynomialRing(QQ, 5, order=T)
sage: print(P.repr_long())
Polynomial Ring
  Base Ring : Rational Field
       Size : 5 Variables
   Block  0 : Ordering : lex
              Names    : a, b, c
   Block  1 : Ordering : degrevlex
              Names    : d, e
sage: a > d^2
True
sage: e > d^2
False
```

Higher Level Operations on Ideals

I.associated_primes, I.complete_primary_decomposition, I.integral_closure, I.intersection, I.dimension, I.is_maximal, I.is_prime, I.is_principal, I.primary_decomposition, I.syzygy_module, I.elimination_ideal ,I.is_trivial, I.radical, I.transformed_basis, I.reduce, I.genus, I.minimal_associated_primes, I.reduced_basis, I.basis_is_groebner, I.groebner_basis, I.multiplicative_order, I.groebner_fan

SymbolicData

The SymbolicData project is set out to develop concepts and tools for testing Computer Algebra Software (CAS) and to collect relevant data from different areas of Computer Algebra. Tools and data are designed to be used both on a local site for special testing purposes and to manage a central repository at www.symbolicdata.org.

```
sage: install_package('database_symbolic_data-20070206')
sage: SD = sage.databases.symbolic_data.SymbolicData()
sage: SD.
Display all 351 possibilities? (y or n)
...
sage: I = SD.Katsura_7
sage: I.gens()[0]
u0^2 + 2*u1^2 + 2*u2^2 + 2*u3^2 + 2*u4^2 + 2*u5^2 + 2*u6^2 + 2*u7^2 - u0
sage: I.ring()
Polynomial Ring in u0, u1, u2, u3, u4, u5, u6, u7 over Rational Field
```

Applications: MQ

We call MQ the problem of finding the common roots of a set of at most quadratic polynomials. Some research in symmetric cryptanalysis recently devoted to expressing ciphers as MQ problems and solving them. SAGE now has a class **MPolynomialSystem** to support this.

```
sage: sr = mq.SR(2,1,1,4)
sage: F,s = sr.polynomial_system()
sage: F
Polynomial System with 72 Polynomials in 36 Variables

sage: gb = F.groebner_basis()
sage: A,v = F.coeff_matrix()
sage: (A*v).list() == F.gens()
True

sage: gb[0]
k002 + (a^2 + a)*k003 + (a^2 + a)

sage: sr = mq.SR(10,4,4,8,star=True) # AES
sage: F,s = sr.polynomial_system()
sage: F
Polynomial System with 8576 Polynomials in 4288 Variables
```

Desperately Needed

Missing Functionality

- decent basic arithmetic for multivariate polynomials over commutative rings ($\mathbb{Z}$, $\mathbb{Z}_n$)
  - SINGULAR (unofficial, preliminary version available)
  - CoCoALib: on todo list
- Gröbner bases over commutative rings
  - Macaulay2: optional SAGE package, can do it
  - SINGULAR (eventually?)
  - CoCoALib: on todo list
- new base fields for fast arithmetic: $\mathbb{Q}(\alpha)$, $\mathbb{R}$, $\mathbb{C}$.
  - $\mathbb{Q}(\alpha)$: SINGULAR can do it
  - ApCoCoALib: should be very good for $\mathbb{R}$, $\mathbb{C}$.

Outline

Echelon Form of a Sparse Matrix over $\mathbb{F}_q$

One would expect that there are several implementations to compute this, but:

MAGMA does not provide a command to do this, also: null space is dense.

LinBox almost does it when computing the rank but kills unneeded rows, to preserve memory. Also, they have another algorithm which involves column swaps to compute the rank

SAGE has a native implementation, which is surprisingly good. "We use Gauss elimination, in a slightly intelligent way, in that we clear each column using a row with the minimum number of nonzero entries."

g0n by John Cremona has an implementation. Gently ripped out by Ralf Weinmann right now and templated.

Some Preliminary Timings for g0n

Corner Case Algorithm

based on William Stein's rational-echelon-via-solve

1. Compute $r = rank(A)$. This is cheaper than Gaussian elimination because we can use "Symbolic Reordering";
2. Compute the pivot columns of the transpose A^t of A via "Symbolic Reordering".
3. Let B be the submatrix of A consisting of the rows corresponding to the pivot columns found in the previous step. Note that, aside from zero rows at the bottom, B and A have the same reduced row echelon form.
4. Compute the pivot columns of B.
5. Let C be the submatrix of B of pivot columns. Let D be the complementary submatrix of B of all all non-pivot columns. Use a solver (such as Wiedemann) to find the matrix X such that $CX = D$. I.e., solve a bunch of linear systems of the form $Cx = v$, where the columns of X are the solutions.
6. Return the matrix $I||X$ where I is the identity matrix of rank r.

Source

```
def echelon_form_via_solve(A):
    r = A.rank() # Step 1: Compute the rank

    if r == self.nrows():
        B = A
    else:
        # Steps 2 and 3: Extract out a submatrix of full rank.
        P = A.transpose().pivots()
        B = A.matrix_from_rows(P)

    # Step 4: Now we instead worry about computing the reduced row echelon form of B.
    pivots = B.pivots()

    # Step 5: Apply solver
    C = B.matrix_from_columns(pivots)
    pivots_ = set(pivots)
    non_pivots = [i for i in range(B.ncols()) if not i in pivots_]
    D = B.matrix_from_columns(non_pivots)
    X = C.solve_right(D, algorithm="LinBox:Blackbox")

    R = self.parent()()
    for i in range(len(pivots)): R[i,pivots[i]] = 1
    for i in range(X.nrows()):
        for j in range(X.ncols()):
            R[i,non_pivots[j]] = X[i,j]
    return R
```

Complexity

Let A be a $m \times n$ matrix over $\mathbb{F}_p$. Furthermore, let r be A's rank and ω the average number of nonzero entries per row. Then the time complexity is:

- We compute the pivot columns twice, each costing $2\sum_{k=1}^{r}(m-k)min(\omega 2^k, n-k)$.
- We solve for $n-r$ vectors w.r.t. to a $r \times r$ matrix, each costing $r^{2+\epsilon}$ field operations.

However, the main feature is that we don't need to write down the intermediate matrices during Gaussian elimination.

Claim

If r is significantly bounded away from $min(m,n)$ and the matrix is not too sparse, this gives better results speed and memory wise than structured Gaussian elimination.

Questions?

Thank You!