Algebraic Structures - Sage+Oscar Days

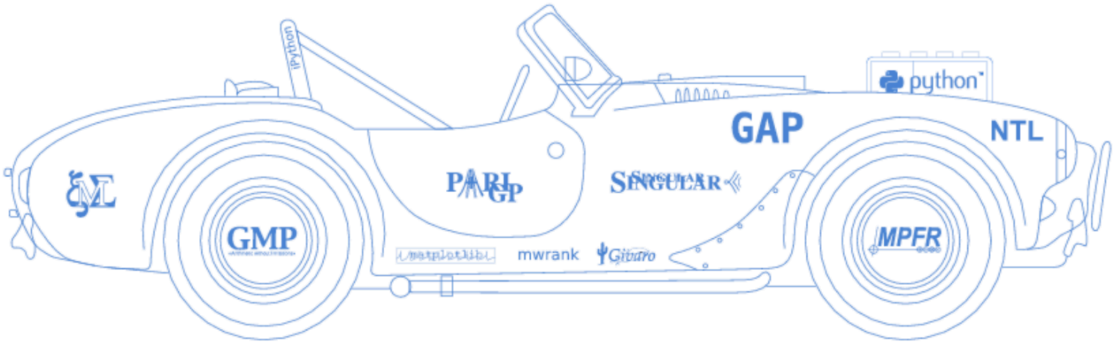
February 15, 2021

1 Algebraic Structures in SageMath

- Building the car: other libraries
- Basic rings: , , ,
- Polynomials and power series
- Parents and coercion
- Finite fields
- And beyond: linear algebra, symbolics, creating new parents

1.1 Building the car: other libraries

Part of the original philosophy of Sage is to take advantage of the best existing open source algebra systems, tying them together through Python.



- Using OSCAR from Sage and vice versa via PyJulia
- Determining libraries used using `get_systems` (requires building Sage with `SAGE_PROFILE=yes`):

[2]:
```
R.<x,y> = ZZ[]
from sage.misc.citation import get_systems
get_systems('x+y+1')
```

[2]:
```
['Singular']
```

- A list of all libraries used by Sage is available here.
- Some systems are used as libraries (e.g. libgap, pari), others by a Pexpect interface (e.g. magma, fricas). Pexpect is slower but easier: passes strings.

1.2 Basic rings

: Integer arithmetic uses GMP (as of `9.3.beta2`), preparses to get `Integer` objects rather than Python ints. Various number theoretic methods.

[5]:
```
(2^512).next_prime()
```

[5]:
```
13407807929942597099574024998205846127479365820592393377723561443721764030073546
976801874298166903427690031858186486050853753882811946569946433649006084171
```

[6]:
```
(2^256+17).factor()
```

[6]:
```
3 * 2549 * 1762903 *
85893318673072077209672370448619845606617549995039365757637791050 33
```

: Rational numbers use GMP's `mpq_t` type.

[7]:
```
bernoulli(70).support()
```

[7]:
```
[2, 3, 5, 7, 11, 71, 688531, 20210499584198062453, 3090850068576441179447]
```

: Several models of real numbers. Python floats, `RDF` using C doubles, MPFR for multi-precision floating point, MPFI for interval arithmetic, Arb for arbitrary precision balls.

[8]:
```
RR.pi()
```

[8]:
```
3.14159265358979
```

[9]:
```
RDF.pi()
```

[9]:
```
3.141592653589793
```

[10]:
```
import math
type(math.pi)
```

[10]:
```
<class 'float'>
```

[11]:
```
RealIntervalField(200).pi()
```

[11]:
```
3.14159265358979323846264338327950288419716939937510582097495?
```

[12]:
```
RealBallField(200).pi()
```

[12]:
```
[3.14159265358979323846264338327950288419716939937510582097494 +/- 5.73e-60]
```

: Multiple models of complex numbers as well. Python complex type, `CDF` using C doubles, MPC for multi-precision, MPFI for rectangles, Arb for balls.

[13]:
```
complex(1,1)^3
```

[13]:
```
(-2+2j)
```

[14]:
```
CDF("I").exp()
```

[14]:
```
0.5403023058681398 + 0.8414709848078965*I
```

[15]:
```
CC("I").exp()
```

[15]:
```
0.540302305868140 + 0.841470984807897*I
```

[17]:
```
ComplexIntervalField(100).gen().exp()
```

[17]:
```
0.54030230586813971740093660744? + 0.84147098480789650665250232163 1?*I
```

[19]:
```
ComplexBallField(100).gen(0).exp()
```

[19]:
```
[0.54030230586813971740093660744 +/- 3.36e-30] +
[0.84147098480789650665250232163 0 +/- 8.45e-31]*I
```

1.3 Polynomials and power series

- Generic code supports arbitrary base rings (both sparse and dense).
- Univariate and multivariate polynomials behave quite differently (term orders for multivariate)
- Libraries used for specific base fields give big speedups and additional functionality
- FLINT, NTL, Arb, MPFR for univariate; Singular, Macaulay2 for multivariate; Polybori for boolean polynomials

[20]:
```
R.<x> = ZZ[] # preparser syntax
(x^18-1).factor()
```

[20]:
```
(x - 1) * (x + 1) * (x^2 - x + 1) * (x^2 + x + 1) * (x^6 - x^3 + 1) * (x^6 + x^3
+ 1)
```

[21]:
```
(x^6 - x^3 + 1).is_cyclotomic()
```

[21]:
```
True
```

[22]:
```
1/(1 - x + x^2 + O(x^10))
```

[22]:
```
1 + x - x^3 - x^4 + x^6 + x^7 - x^9 + O(x^10)
```

[39]:
```
S.<x,y> = PolynomialRing(QQ, order='degrevlex')
I = S.ideal([-3*x^2 - x*y - y^2 + y, -x^2 + x*y - y^2 - 9*x])
sorted(I.groebner_basis())
```

[39]:
```
[x*y - 1/2*y^2 - 27/4*x - 1/4*y,
 x^2 + 1/2*y^2 + 9/4*x - 1/4*y,
 y^3 - 3/4*y^2 + 261/8*x + 101/24*y]
```

[41]:
```
T.<x,y> = PolynomialRing(QQ, order='lex')
I = T.ideal([-3*x^2 - x*y - y^2 + y, -x^2 + x*y - y^2 - 9*x])
sorted(I.groebner_basis())
```

[41]:
```
[y^4 - 15/2*y^3 + 307/12*y^2 - 81/4*y, x + 8/261*y^3 - 2/87*y^2 + 101/783*y]
```

1.4 Parents and coercion

- Many objects in Sage fall into one of three levels: an element (e.g. a polynomial), a parent (e.g. the ring) or a category (e.g. commutative rings with unity).
- A parent is simultaneously a set of elements and an object in a specific category.
- The integer 2 and the rational number 2/1 have different parents but we want to consider them equal.
- Solution: create a commuting family of homomorphisms between parents in Sage, called coercions.

[42]:
```
f = QQ.coerce_map_from(ZZ)
f
```

[42]:
```
Natural morphism:
  From: Integer Ring
  To:   Rational Field
```

[44]:
```
f(2), f(2).parent()
```

[44]:
```
(2, Rational Field)
```

[45]:
```
2 + (1/2)
```

[45]:
```
5/2
```

- Most coercions are injective, but some are quotient maps.
- Note that coercions only depend on the parent and should be homomorphisms (in an appropriate category).

[52]:
```
Zmod(12).coerce_map_from(ZZ)
```

[52]:
```
Natural morphism:
  From: Integer Ring
  To:   Ring of integers modulo 12
```

- Sometimes there is no coercion map in either direction but there is a natural pushout: another parent that both coerce to.

[53]:
```
R = PolynomialRing(QQ, 'x')
S = PolynomialRing(ZZ, 'x,y')
print(R.has_coerce_map_from(S))
print(S.has_coerce_map_from(R))
```

```
False
False
```

[54]:
```
x = R.gen(0)
y = S.gen(1)
(x+y).parent()
```

[54]:
```
Multivariate Polynomial Ring in x, y over Rational Field
```

[57]:
```
from sage.categories.pushout import pushout
pushout(R, S)
```

[57]:
```
Multivariate Polynomial Ring in x, y over Rational Field
```

- Pushouts are determined using the `construction()` method on each parent.

[58]:
```
R.construction()
```

[58]:
```
(Poly[x], Rational Field)
```

[59]:
```
S.construction()
```

[59]:
```
(MPoly[x,y], Integer Ring)
```

1.5 Finite fields

- Sage uses six different implementations of finite fields, depending on q.

[73]:
```
# prime fields, q < sqrt(2^31-1)
type(GF(5)(1))
```

[73]:
```
<class 'sage.rings.finite_rings.integer_mod.IntegerMod_int'>
```

[74]:
```
# prime fields, sqrt(2^31-1) < q < 2^32
type(GF(next_prime(2^16))(1))
```

[74]:
```
<class 'sage.rings.finite_rings.integer_mod.IntegerMod_int64'>
```

[75]:
```
# prime fields, q > 2^32
type(GF(next_prime(2^32))(1))
```

[75]:
```
<class 'sage.rings.finite_rings.integer_mod.IntegerMod_gmp'>
```

[79]:
```
# q non-prime, q < 2^16
type(GF(2^15)(1))
```

[79]:
```
<class 'sage.rings.finite_rings.element_givaro.FiniteField_givaroElement'>
```

[80]:
```
# q >= 2^16, power of 2
type(GF(2^16)(1))
```

[80]:
```
<class 'sage.rings.finite_rings.element_ntl_gf2e.FiniteField_ntl_gf2eElement'>
```

[81]:
```
# q odd non-prime, q > 2^16
type(GF(3^16)(1))
```

[81]:
```
<class
'sage.rings.finite_rings.element_pari_ffelt.FiniteFieldElement_pari_ffelt'>
```

When q is not prime, there are different methods for choosing the modulus since there are conflicting requirements.

- We want the computation of the modulus to be fast. Good approach: Adleman-Lenstra algorithm.
- We want maps between fields of the same cardinality to be fast. Good approach: if b generator for p^n and m|n ask that $b^{((p^n-1)/(p^m-1))}$ satisfies modulus for p^m. To make unique, choose lexicographically least polynomials (Conway polynomial)

Sage decides which to prioritize based on whether you specify a **variable name**

[82]:
```
# If q is small enough to be contained in the database of Conway polynomials then
# you'll get a Conway polynomial either way (unless you specify an algorithm)
GF(3^12).modulus()
```

[82]:
```
x^12 + x^6 + x^5 + x^4 + x^2 + 2
```

[84]:
```
(GF(3^24).gen()^((3^24-1)/(3^12-1))).minimal_polynomial()
```

[84]:
```
x^12 + x^6 + x^5 + x^4 + x^2 + 2
```

[85]:
```
# Above the cutoff, if you give a variable name it prioritizes finding the␣
 ↪modulus quickly
GF(997^12, 'a').modulus()
```

[85]:
```
x^12 + 7*x^11 + 15*x^10 + x^9 + 972*x^8 + 24*x^7 + 152*x^6 + 208*x^5 + 184*x^4 +
144*x^3 + 78*x^2 + 51*x + 71
```

[86]:
```
# If you don't give a variable name, it chooses compatibly with divisors
k = GF(997^12)
k.modulus()
```

[86]:
```
x^12 + 721*x^11 + 212*x^10 + 234*x^9 + 9*x^8 + 81*x^7 + 224*x^6 + 631*x^5 +
378*x^4 + 830*x^3 + 916*x^2 + 133*x + 7
```

[87]:
```
(k.gen()^((997^12-1)/(997^6-1))).minimal_polynomial()
```

[87]:
```
x^6 + 981*x^3 + 58*x^2 + 260*x + 7
```

[88]:
```
from sage.databases.conway import ConwayPolynomials
C = ConwayPolynomials()
GF(997)['x'](C[997,6])
```

[88]:
```
x^6 + 981*x^3 + 58*x^2 + 260*x + 7
```

If you will be working outside the regime where Conway polynomials are known, choosing polynomials compatibly can get quite slow. If you can choose a large finite field in advance, it should be faster to fix a modulus there and then construct subfields.

[91]:
```
k.<a> = GF(3^240)
```

[92]:
```
S = k.subfields()
```

```
␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-92-86b9f5197bd8> in <module>
----> 1 S = k.subfields()


~/sage/sage-9.2.beta12/local/lib/python3.8/site-packages/sage/rings/
↪finite_rings/finite_field_base.pyx in sage.rings.finite_rings.
↪finite_field_base.FiniteField.subfields (build/cythonized/sage/rings/
↪finite_rings/finite_field_base.c:17445)()
   1559         pairs = []
   1560         for m in divisors:
-> 1561             K = self.subfield(m, name=name[m])
   1562             pairs.append((K, self.coerce_map_from(K)))
   1563         return pairs


~/sage/sage-9.2.beta12/local/lib/python3.8/site-packages/sage/rings/
↪finite_rings/finite_field_base.pyx in sage.rings.finite_rings.
↪finite_field_base.FiniteField.subfield (build/cythonized/sage/rings/
↪finite_rings/finite_field_base.c:16728)()
   1472         else:
   1473             gen = self.gen()**((self.order() - 1)//(p**degree - 1))
```

```
   -> 1474             K = GF(p**degree, modulus=gen.minimal_polynomial(),␣
↪name=name)
      1475             try: # to register a coercion map, embedding of K to self
      1476                 self.register_coercion(K.hom([gen], codomain=self,␣
↪check=False))


      ~/sage/sage-9.2.beta12/local/lib/python3.8/site-packages/sage/structure/
↪factory.pyx in sage.structure.factory.UniqueFactory.__call__ (build/cythonized/
↪sage/structure/factory.c:2181)()
      365             False
      366         """
   --> 367         key, kwds = self.create_key_and_extra_args(*args, **kwds)
      368         version = self.get_version(sage_version)
      369         return self.get_object(version, key, kwds)


      ~/sage/sage-9.2.beta12/local/lib/python3.8/site-packages/sage/rings/
↪finite_rings/finite_field_constructor.py in create_key_and_extra_args(self,␣
↪order, name, modulus, names, impl, proof, check_irreducible, prefix, repr,␣
↪elem_cache, base, **kwds)
      659             if modulus is not None:
      660                 if modulus.degree() != relative_degree:
   --> 661                     raise ValueError("degree of the modulus does not␣
↪match the relative degree of finite field")
      662                 if check_irreducible and not modulus.is_irreducible():
      663                     raise ValueError("finite field modulus must be␣
↪irreducible but it is not")


      ValueError: degree of the modulus does not match the relative degree of␣
↪finite field
```

For number fields, one can construct relative extensions. Applications include finding minimal polynomials, norms and traces in this relative extension. Work is in progress to add relative extensions of finite fields.

Obviously there are many more algebraic structures in Sage.

1.5.1 Linear algebra

You can work with matrices or modules

[93]:
```
M = matrix(3,3,range(9))
```

[94]:
```
M.kernel()
```

[94]:
```
Free module of degree 3 and rank 1 over Integer Ring
Echelon basis matrix:
[ 1 -2  1]
```

[95]:
```
M.rref()
```

[95]:
```
[ 1  0 -1]
[ 0  1  2]
[ 0  0  0]
```

[101]:
```
V = QQ^3
W = V.subspace([[1,2,3], [4,5,6]])
W
```

[101]:
```
Vector space of degree 3 and dimension 2 over Rational Field
Basis matrix:
[ 1  0 -1]
[ 0  1  2]
```

[105]:
```
X = V.subspace([[1,2,1],[7,8,9]])
X.intersection(W)
```

[105]:
```
Vector space of degree 3 and dimension 1 over Rational Field
Basis matrix:
[  1 8/7 9/7]
```

1.6 Symbolics

Symbolics (symbolic functions, integration and differentiation, solving equations) is mostly separated from the rings and fields in Sage (though the "symbolic ring" is a parent).

One warning: `x` is predefined in Sage as a symbolic variable, and you can also get symbolics by taking roots of integers for example. Unless you're doing calculus or solving symbolic equations, you should usually try to avoid the symbolic ring, since it's much slower than polynomial rings for arithmetic.

[107]:
```
x = var('x')
f = e^x*cos(x)
f.derivative(x)
```

[107]:
```
cos(x)*e^x - e^x*sin(x)
```

[108]:
```
f.integral(x)
```

[108]:
```
1/2*(cos(x) + sin(x))*e^x
```

[109]:
```
f.integrate(x, 0, pi)
```

[109]:
```
-1/2*e^pi - 1/2
```

1.7 Creating parents

When you need an algebraic structure not yet in Sage, you'll generally need to implement a parent class and an element class. See Simon King's guide

[ ]: