Infrastructure for generic code in SageMath: categories, axioms, constructions

Nicolas M. Thiéry

LRI, Université Paris Sud 11

July 28th of 2016, CICM, Białystok
Abstract

The SageMath systems provides thousands of mathematical objects and tens of thousands of operations to compute with them. A system of this scale requires an infrastructure for writing and structuring generic code, documentation, and tests that apply uniformly on all objects within certain realms.

In this talk, we describe the infrastructure implemented in SageMath. It is based on the standard object oriented features of Python, together with mechanisms to scale (dynamic classes, mixins, ...) thanks to the rich available semantic (categories, axioms, constructions). We relate the approach taken with that in other systems, and discuss work in progress.
SageMath: a general purpose software for mathematics

Numbers: \(42, \frac{7}{9}, \frac{1+\sqrt{3}}{2}, \pi, 2.71828182845904523536028747?\)

Matrices:
\[
\begin{pmatrix}
4 & -1 & 1 & -1 \\
-1 & 2 & -1 & -1 \\
0 & 5 & 1 & 3
\end{pmatrix},
\begin{pmatrix}
1.000 & 0.500 & 0.333 \\
0.500 & 0.333 & 0.250 \\
0.333 & 0.250 & 0.200
\end{pmatrix}
\]

Polynomials: \(-9x^8 + x^7 + x^6 - 13x^5 - x^3 - 3x^2 - 8x + 4\)

Series: \(1 + 1x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \cdots\)

Symbolic expressions, equations: \(\cos(x)^2 + \sin(x)^2 = 1\)

Finite fields, algebraic extensions, elliptic curves, ...
**SageMath**: a general purpose software for mathematics

**Numbers**: $42, \frac{7}{9}, \frac{1+\sqrt{3}}{2}, \pi, 2.71828182845904523536028747?$

Matrices:

$$
\begin{pmatrix}
4 & -1 & 1 & -1 \\
-1 & 2 & -1 & -1 \\
0 & 5 & 1 & 3
\end{pmatrix},
\begin{pmatrix}
1.000 & 0.500 & 0.333 \\
0.500 & 0.333 & 0.250 \\
0.333 & 0.250 & 0.200
\end{pmatrix}
$$

**Polynomials**: $-9x^8 + x^7 + x^6 - 13x^5 - x^3 - 3x^2 - 8x + 4$

**Series**: $1 + 1x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \cdots$

**Symbolic expressions, equations**: $\cos(x)^2 + \sin(x)^2 = 1$

**Finite fields, algebraic extensions, elliptic curves**, ...
SageMath: a general purpose software for mathematics

Numbers: 42, $\frac{7}{9}$, $\frac{1+\sqrt{3}}{2}$, $\pi$, 2.71828182845904523536028747?

Matrices:
$$
\begin{pmatrix}
4 & -1 & 1 & -1 \\
-1 & 2 & -1 & -1 \\
0 & 5 & 1 & 3
\end{pmatrix},
\begin{pmatrix}
1.000 & 0.500 & 0.333 \\
0.500 & 0.333 & 0.250 \\
0.333 & 0.250 & 0.200
\end{pmatrix}
$$

Polynomials: $-9x^8 + x^7 + x^6 - 13x^5 - x^3 - 3x^2 - 8x + 4$

Series: $1 + 1x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \cdots$

Symbolic expressions, equations: $\cos(x)^2 + \sin(x)^2 = 1$

Finite fields, algebraic extensions, elliptic curves, ...
SageMath: a general purpose software for mathematics

Numbers: $42, \frac{7}{9}, \frac{1+\sqrt{3}}{2}, \pi, 2.71828182845904523536028747$?

Matrices:
\[
\begin{pmatrix}
4 & -1 & 1 & -1 \\
-1 & 2 & -1 & -1 \\
0 & 5 & 1 & 3
\end{pmatrix}
, \quad \begin{pmatrix}
1.000 & 0.500 & 0.333 \\
0.500 & 0.333 & 0.250 \\
0.333 & 0.250 & 0.200
\end{pmatrix}
\]

Polynomials: $-9x^8 + x^7 + x^6 - 13x^5 - x^3 - 3x^2 - 8x + 4$

Series: $1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \cdots$

Symbolic expressions, equations: $\cos(x)^2 + \sin(x)^2 = 1$

Finite fields, algebraic extensions, elliptic curves, ...
SageMath: a general purpose software for mathematics

Numbers: 42, \( \frac{7}{9}, \frac{1+\sqrt{3}}{2}, \pi, 2.71828182845904523536028747 \)

Matrices:
\[
\begin{pmatrix}
4 & -1 & 1 & -1 \\
-1 & 2 & -1 & -1 \\
0 & 5 & 1 & 3
\end{pmatrix},
\begin{pmatrix}
1.000 & 0.500 & 0.333 \\
0.500 & 0.333 & 0.250 \\
0.333 & 0.250 & 0.200
\end{pmatrix}
\]

Polynomials: \(-9x^8 + x^7 + x^6 - 13x^5 - x^3 - 3x^2 - 8x + 4\)

Series: \(1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \cdots\)

Symbolic expressions, equations: \(\cos(x)^2 + \sin(x)^2 = 1\)

Finite fields, algebraic extensions, elliptic curves, ...
Combinatorial objects

\[
\begin{array}{cccc}
1 & 3 & 4 & 7 \\
2 & 5 & 6 & 8
\end{array}
\]

\[
\frac{1}{6}q^2 - \frac{1}{6}q \quad \frac{q^2}{q^5 + 2q^4 + 3q^3 + 3q^2 + 2q + 1} \quad \frac{1}{2}q \quad \frac{1}{q^4 + q^3 + 2q^2 + q + 1}
\]

\[
01001010010010100100010100100100101001010010\cdots
\]
Graphs
Geometric objects
Sage : a large library of mathematical objects and algorithms

- 1.5M lines of code/doc/tests (Python/Cython) + dependencies
- 1k+ types of objects
- 2k+ methods and functions
- 200 regular contributors

Problems

- How to structure this library
- How to guide the user
- How to promote consistency and robustness?
- How to reduce duplication?
Sage: a large library of mathematical objects and algorithms

- 1.5M lines of code/doc/test (Python/Cython) + dependencies
- 1k+ types of objects
- 2k+ methods and functions
- 200 regular contributors

Problems

- How to structure this library
- How to guide the user
- How to promote consistency and robustness?
- How to reduce duplication?
Example: binary powering

\begin{verbatim}
sage: m = 3
sage: m^8 == m*m*m*m*m*m*m*m == ((m^2)^2)^2
  True

sage: m = random_matrix(QQ, 4)
sage: m^8 == m*m*m*m*m*m*m*m == ((m^2)^2)^2
  True
\end{verbatim}

- Complexity: \( O(\log(k)) \) instead of \( O(k) \)!
- We would want a single generic implementation!
Example: binary powering

```
sage: m = 3
sage: m^8 == m*m*m*m*m*m*m*m == ((m^2)^2)^2
   True

sage: m = random_matrix(QQ, 4)
sage: m^8 == m*m*m*m*m*m*m*m == ((m^2)^2)^2
   True
```

- Complexity: $O(\log(k))$ instead of $O(k)$!
- We would want a single *generic* implementation!
Example: binary powering II

Algebraic realm

- **Semigroup**: a set $S$ endowed with an associative binary internal law $*$
- The integers form a semigroup
- Square matrices form a semigroup

We want to

- Implement $\text{pow}\_\text{exp}(x,k)$
- Specify that
  - if $x$ is an *element* of a semigroup
  - then $x^k$ can be computed with $\text{pow}\_\text{exp}(x,k)$

What happens if

- $x$ is an element of a group? of a finite group?
Example: binary powering II

Algebraic realm

- **Semigroup**: a set $S$ endowed with an associative binary internal law $*$
- The integers form a semigroup
- Square matrices form a semigroup

We want to

- Implement `pow_exp(x, k)`
- Specify that
  - if $x$ is an *element* of a semigroup
  - then $x^k$ can be computed with `pow_exp(x, k)`

What happens if

- $x$ is an element of a group? of a finite group?
Example: binary powering II

Algebraic realm

- **Semigroup**: a set $S$ endowed with an associative binary internal law $*$
- The integers form a semigroup
- Square matrices form a semigroup

We want to

- Implement $\text{pow\_exp}(x, k)$
- Specify that
  - if $x$ is an *element* of a semigroup
  - then $x^k$ can be computed with $\text{pow\_exp}(x, k)$

What happens if

- $x$ is an element of a group? of a finite group?
**Selection mechanism**

**We want**

- Design a *hierarchy of realms* and specify the operations there
- Provide generic implementations of those operations
- Specify in which realm they are valid
- Specify in which realm each object is

**We need a *selection mechanism*:**

- to resolve the call $f(x)$
- by selecting the most specific implementation of $f$
Selection mechanism

We want

- Design a **hierarchy of realms** and specify the operations there
- Provide generic implementations of those operations
- Specify in which realm they are valid
- Specify in which realm each object is

We need a **selection mechanism**:

- to resolve the call \( f(x) \)
- by selecting the most specific implementation of \( f \)
Designing a hierarchy of realms for mathematics

In general
Hard problem: isolate the proper business concepts

In mathematics

- “Few” fundamental concepts:
  - basic operations/structure: ∈, +, *, cardinality, topology, ...
  - axioms: associative, finite, compact, ...
  - constructions: cartesian product, quotients, ...

- Concepts known by the users

- All the richness comes from combining those few concepts to form many realms:
  groups, fields, semirings, lie algebras, ...
Designing a hierarchy of realms for mathematics

In general
Hard problem: isolate the proper business concepts

In mathematics

- “Few” fundamental concepts:
  - basic operations/structure: ∈, +, *, cardinality, topology, ...
  - axioms: associative, finite, compact, ...
  - constructions: cartesian product, quotients, ...

- Concepts known by the users
- All the richness comes from combining those few concepts to form many realms:
  groups, fields, semirings, lie algebras, ...
A hierarchy of realms based on mathematical categories

A robust hierarchy based on a century of abstract algebra
Axiom, Aldor, MuPAD

- Specific language
- Selection mechanism: “object oriented programming”
- Hierarchy of “abstract classes” modeling the mathematical categories

Example

category Semigroups :
category Magmas ;

intpow := proc(x, k) ... 
// other methods
Pioneers 1980- II

GAP

- Specific language
- One filter per fundamental concept:
  IsMagma(G), IsAssociative(G), ...
- InstallMethod(Operation, filters, method)
- Method selection according to the filters that are know to be satisfied by $\times$
- Implicit modeling of the hierarchy

Example

```
powExp := function(n, k) ...
InstallMethod(pow, [IsMagma, IsAssociative], powExp)
```
Related developments

Focal (Certified CAS)
  • Species

MathComp (Proof assistant)
  • Canonical structures

MMT (Knowledge management)
  • E.g. LATIN’s theories
Strategical choices

- A standard language (Python)
- Selection mechanism: object oriented programming

Specific features

- Distinction Element/Parent (as in Magma)
- Morphisms
- Functorial constructions
- Axioms

Constraints

- Partial compilation (Cython), serialization
- Multiple inheritance with Python / Cython
- Scaling!
Implementation in Sage (2008-)

Strategical choices

- A standard language (Python)
- Selection mechanism: object oriented programming

Specific features

- Distinction Element/Parent (as in Magma)
- Morphisms
- Functorial constructions
- Axioms

Constraints

- Partial compilation (Cython), serialization
- Multiple inheritance with Python / Cython
- Scaling!
Implementation in Sage (2008-)

Strategical choices

- A standard language (Python)
- Selection mechanism: object oriented programming

Specific features

- Distinction Element/Parent (as in Magma)
- Morphisms
- Functorial constructions
- Axioms

Constraints

- Partial compilation (Cython), serialization
- Multiple inheritance with Python / Cython
- Scaling!
The standard Python Object Oriented approach

Abstract classes for elements

class MagmaElement:
    @abstract_method
def __mul__(x,y):

class SemigroupElement(MagmaElement):
def __pow__(x,k): ...

A concrete class

class MySemigroupElement(SemigroupElement):
    # Constructor, data structure, ...
def __mul__(x,k): ...
Standard OO: classes for parents

Abstract classes

```python
class Semigroup(Magma):
    @abstract_method
    def semigroup_generators(self):
    def cayley_graph(self):
```

A concrete class

```python
class MySemigroup(Semigroup):
    def semigroup_generators(self):
```
Standard OO : hierarchy of abstract classes

```python
class Set: ... 
class SetElement: ... 
class SetMorphism: ... 

class Magma (Set): ... 
class MagmaElement (SetElement): ... 
class MagmaMorphism (SetMorphism): ... 

class Semigroup (Magma): ... 
class SemigroupElement (MagmaElement): ... 
    def __pow__(self, k): ... 
class SemigroupMorphism (MagmaMorphism): ... 
```

Hmm, this code smells, doesn’t it?

- How to avoid duplication?
Standard OO: hierarchy of abstract classes

class Set: ...
class SetElement: ...
class SetMorphism: ...

class Magma (Set): ...
class MagmaElement (SetElement): ...
class MagmaMorphism(SetMorphism): ...

class Semigroup (Magma): ...
class SemigroupElement (MagmaElement): ...
    def __pow__(self, k): ...
class SemigroupMorphism(MagmaMorphism): ...

Hmm, this code smells, doesn’t it?

• How to avoid duplication?
Sage’s approach: categories and mixin classes

Categories

class Semigroups(Category):
    def super_categories():
        return [Magmas()]
class ParentMethods:
    class ElementMethods:
        def __pow__(x, k):
    class MorphismMethods:

class MySemigroup(Parent):
    def __init__(self):
        Parent.__init__(self, category=Semigroups())
    def semigroup_generators(self):
class Element:
    # constructor, data structure
    def __mul__(x, y):...
Usage

```python
sage : S = MySemigroup()
sage : S.category()
Category of semigroups
sage : S.cayley_graph()

sage : S.__class__.__mro()
[<class 'MySemigroup_with_category'>, ...
 <type 'sage.structure.parent.Parent'>, ...
 <class 'Semigroups.parent_class'>,
 <class 'Magmas.parent_class'>,
 <class 'Sets.parent_class'>, ...]
```

Generic tests

```python
sage : TestSuite(S).run( verbose=True)
...
running ._test_associativity() . . . pass
running ._test_cardinality() . . . pass
running ._test_elements_eq_transitive() . . . pass
```
Usage

```python
sage : S = MySemigroup()
sage : S.category()
Category of semigroups
sage : S.cayley_graph()
```

```python
sage : S.__class__.__mro()
[<class 'MySemigroup_with_category'>, ... 
 <type 'sage.structure.parent.Parent'>, ... 
 <class 'Semigroups.parent_class'>,
 <class 'Magmas.parent_class'>,
 <class 'Sets.parent_class'>, ...]
```

Generic tests

```python
sage : TestSuite(S).run(verbose=True)
...
running ._test_associativity() ... pass
running ._test_cardinality() ... pass
running ._test_elements_eq_transitive() ... pass
```
How does this work?

Dynamic construction, from the mixins, of:

- three hierarchies of abstract classes:
  - set
    - magma
      - unital magma
        - inverse unital magma
          - group
    - semigroup
  - set element
    - magma element
      - unital magma element
        - inverse unital magma element
          - monoid element
            - group element
    - semigroup element
  - set morphism
    - magma morphism
      - unital magma morphism
        - inverse unital magma morphism
          - monoid morphism
            - group morphism

- the concrete classes for parents and elements
Summary

Explicit modeling of

- Elements, Parents, Morphisms, Homsets
- Categories: bookshelves about a given realm:
  - Semantic information
  - Mixins for parents, elements, morphisms, homsets:
    Generic Code, Documentation, Tests

Method selection mechanism

- Standard Object Oriented approach
- With a twist: classes constructed dynamically from mixins

Isn’t this gross overdesign?

- Deviation from standard Python, additional complexity
- Higher learning curve
Summary

Explicit modeling of

- Elements, Parents, Morphisms, Homsets
- Categories: bookshelves about a given realm:
  - Semantic information
  - Mixins for parents, elements, morphisms, homsets:
    Generic Code, Documentation, Tests

Method selection mechanism

- Standard Object Oriented approach
- With a twist: classes constructed dynamically from mixins

Isn’t this gross overdesign?

- Deviation from standard Python, additional complexity
- Higher learning curve
Summary

Explicit modeling of

- Elements, Parents, Morphisms, Homsets
- Categories: *bookshelves* about a given realm:
  - Semantic information
  - Mixins for parents, elements, morphisms, homsets: Generic Code, Documentation, Tests

Method selection mechanism

- Standard Object Oriented approach
- With a twist: classes constructed dynamically from mixins

Isn’t this gross overdesign?

- Deviation from standard Python, additional complexity
- Higher learning curve
Summary

Explicit modeling of

- Elements, Parents, Morphisms, Homsets
- Categories: *bookshelves* about a given realm:
  - Semantic information
  - Mixins for parents, elements, morphisms, homsets: Generic Code, Documentation, Tests

Method selection mechanism

- Standard Object Oriented approach
- *With a twist*: classes constructed dynamically from mixins

Isn’t this gross overdesign?

- Deviation from standard Python, additional complexity
- Higher learning curve
It’s all about scaling

```python
sage : GF3 = mygap.GF(3)
sage : C = cartesian_product([ZZ, RR, GF3])

sage : c = C.an_element(); c
(1, 1.00000000000000000000000000000000, 0*Z(3))
sage : (c+c)^3
(8, 8.00000000000000000000000000000000, 0*Z(3))

sage : C.category()
Category of Cartesian products of commutative rings

sage : C.category().super_categories()
[Category of commutative rings,
  Category of Cartesian products of distributive magmas and additive magmas,
  Category of Cartesian products of monoids,
  Category of Cartesian products of commutative magmas,
  Category of Cartesian products of commutative additive groups]
sage : len(C.categories())
44
```
It’s all about scaling

```
sage : GF3 = mygap.GF(3)
sage : C = cartesian_product([ZZ, RR, GF3])

sage : c = C.an_element(); c
(1, 1.000000000000000000000000, 0*Z(3))
sage : (c+c)^3
(8, 8.000000000000000000000000, 0*Z(3))

sage : C.category()
Category of Cartesian products of commutative rings

sage : C.category().super_categories()
[Category of commutative rings,
 Category of Cartesian products of distributive magmas and additive magmas,
 Category of Cartesian products of monoids,
 Category of Cartesian products of commutative magmas,
 Category of Cartesian products of commutative additive groups]
```
It’s all about scaling

```sage
sage : GF3 = mygap.GF(3)
sage : C = cartesian_product([ZZ, RR, GF3])

sage : c = C.an_element(); c
    (1, 1.0000000000000000, 0*Z(3))
sage : (c+c)^3
    (8, 8.0000000000000000, 0*Z(3))

sage : C.category()
Category of Cartesian products of commutative rings

sage : C.category().super_categories()
[Category of commutative rings,
  Category of Cartesian products of distributive magmas and additive magmas,
  Category of Cartesian products of monoids,
  Category of Cartesian products of commutative magmas,
  Category of Cartesian products of commutative additive groups]
```

```sage
sage : len(C.categories())
44
```
It’s all about scaling

```
sage : GF3 = mygap.GF(3)
sage : C = cartesian_product([ZZ, RR, GF3])

sage : c = C.an_element(); c
(1, 1.00000000000000, 0*Z(3))
sage : (c+c)^3
(8, 8.00000000000000, 0*Z(3))

sage : C.category()
Category of Cartesian products of commutative rings

sage : C.category().super_categories()
[Category of commutative rings,
 Category of Cartesian products of distributive magmas and additive magmas,
 Category of Cartesian products of monoids,
 Category of Cartesian products of commutative magmas,
 Category of Cartesian products of commutative additive groups]
sage : len(C.categories())
44
```
Taming the combinatorial explosion

Categories for groups:

- sets
- magmas
- unital magmas
- inverse unital magmas
- groups
- monoids
- semigroups
Taming the combinatorial explosion

Categories for finite groups:

- sets
- finite sets
- magmas
- semigroups
- unital magmas
- monoids
- inverse unital magmas
- groups
- finite monoids
- finite semigroups
- finite sets
- finite groups

Implemented categories: 11 out of 14
Explicit inheritance: 1 + 9 out of 15
Taming the combinatorial explosion

Categories for finite groups:

- Sets
- Magmas
  - Associative
  - Unital
- Semigroups
  - Unital
- Monoids
  - Inverse
- Groups
  - Finite sets
  - Finite semigroups
  - Finite monoids
  - Finite groups

Implemented categories: 11 out of 14
Explicit inheritance: 1 + 9 out of 15
Taming the combinatorial explosion

Categories for finite groups:

Implemented categories: 11 out of 14
Explicit inheritance: 1 + 9 out of 15
Taming the combinatorial explosion

Categories for finitely generated finite commutative groups:

Implemented categories: 17 out of $\approx 54$
Explicit inheritance: 1 + 15 out of 32
Taming the combinatorial explosion

Categories for finitely generated finite commutative groups:

Implemented categories: 17 out of $\approx 54$
Explicit inheritance: $1 + 15$ out of 32
Taming the combinatorial explosion

Categories for finitely generated finite commutative groups:

Implemented categories: 17 out of $\approx 54$
Explicit inheritance: $1 + 15$ out of 32
Taming the combinatorial explosion

Categories for finitely generated finite commutative groups:

Implemented categories: 17 out of \( \approx 54 \)
Explicit inheritance: 1 + 15 out of 32
Taming the combinatorial explosion

All implemented categories for fields:

Implemented categories: 71 out of \( \approx 2^{13} \)
Explicit inheritance: 3 + 64 out of 121
Taming the combinatorial explosion

All categories:

Categories: 265 out of $\approx 2^{50}$
Explicit inheritance: 70 out of 471
The hierarchy of categories as a lattice

The Birkhoff representation theorem states that an element of a distributive lattice can be represented as the meet of the meet-irreducible elements above it.

- $\wedge$: objects in common
  
  `sage : Groups() & Sets().Finite()`
  
  Category of finite groups

- $\vee$: structure in common
  
  `sage : Fields() | Groups()`
  
  Category of monoids
The hierarchy of categories as a lattice

- $\wedge$: objects in common
  
  sage : Groups() & Sets().Finite()
  
  Category of finite groups

- $\vee$: structure in common
  
  sage : Fields() | Groups()
  
  Category of monoids

Birkhoff representation theorem

An element of a distributive lattice can be represented as the meet of the meet-irreducible elements above it.
The hierarchy of categories as distributive a lattice

- △: objects in common

  sage: Groups() & Sets().Finite()
  Category of finite groups

- ∨: structure in common

  sage: Fields() | Groups()
  Category of monoids

Birkhoff representation theorem
An element of a distributive lattice can be represented as the meet of the meet-irreducible elements above it
The hierarchy of categories as distributive a lattice

- $\wedge$: objects in common
  
  ```python
  sage: Groups() & Sets().Finite()
  Category of finite groups
  ```

- $\vee$: structure in common
  
  ```python
  sage: Fields() | Groups()
  Category of monoids
  ```

**Birkhoff representation theorem**

An element of a distributive lattice can be represented as the meet of the meet-irreducible elements above it
The distributive lattice of categories

Basic concepts (meet-irreducible elements)

- 65 structure categories: Magmas, MetricSpaces, Posets, ...
- 34 axioms: Associative, Finite, NoZeroDivisors, Smooth, ...
- 13 constructions: CartesianProduct, Topological, Homsets, ...

```sage
sage : Groups().structure()
frozenset({Category of unital magmas,
            Category of magmas,
            Category of sets with partial maps,
            Category of sets})
sage : Groups().axioms()
frozenset({'Associative', 'Inverse', 'Unital'})
```

Exponentially many potential combinations thereof

```sage
sage : Magmas().Associative() & Magmas().Unital().Inverse()
Category of groups
```
The distributive lattice of categories

Basic concepts (meet-irreducible elements)

- 65 structure categories: Magmas, MetricSpaces, Posets, ...
- 34 axioms: Associative, Finite, NoZeroDivisors, Smooth, ...
- 13 constructions: CartesianProduct, Topological, Homsets, ...

```python
sage: Groups().structure()
frozenseq({Category of unital magmas,
            Category of magmas,
            Category of sets with partial maps,
            Category of sets})
```

```python
sage: Groups().axioms()
frozenseq({'Associative', 'Inverse', 'Unital'})
```

Exponentially many potential combinations thereof

```python
sage: Magmas().Associative() & Magmas().Unital().Inverse()
Category of groups
```
The distributive lattice of categories

Basic concepts (meet-irreducible elements)

- 65 structure categories: Magmas, MetricSpaces, Posets, ...
- 34 axioms: Associative, Finite, NoZeroDivisors, Smooth, ...
- 13 constructions: CartesianProduct, Topological, Homsets, ...

`sage : Groups().structure()`

```python
frozenset({Category of unital magmas,
            Category of magmas,
            Category of sets with partial maps,
            Category of sets})
```

`sage : Groups().axioms()`

```python
frozenset({'Associative', 'Inverse', 'Unital'})
```

Exponentially many potential combinations thereof

```python
sage : Magmas().Associative() & Magmas().Unital().Inverse()
```

Category of groups
Some more examples

```
sage : Mul = Magmas().Associative().Unital()
Category of monoids

sage : Add = AdditiveMagmas().AdditiveAssociative().AdditiveCommutative().AdditiveUnital()
Category of commutative additive monoids

sage : (Add & Mul).Distributive()
Category of semirings

sage : _.AdditiveInverse()
Category of rings

sage : _.Division()
Category of division rings

sage : _ & Sets().Finite()
Category of finite fields
```
Some more examples

```
sage : Mul = Magmas().Associative().Unital()
Category of monoids

sage : Add = AdditiveMagmas().AdditiveAssociative().AdditiveCommutative().AdditiveUnital()
Category of commutative additive monoids

sage : (Add & Mul).Distributive()
Category of semirings

sage : _.AdditiveInverse()
Category of rings

sage : _.Division()
Category of division rings

sage : _ & Sets().Finite()
Category of finite fields
```
Some more examples

```
sage : Mul = Magmas().Associative().Unital()
Category of monoids

sage : Add = AdditiveMagmas().AdditiveAssociative().AdditiveCommutative().AdditiveUnital()
Category of commutative additive monoids

sage : (Add & Mul).Distributive()
Category of semirings

sage : _.AdditiveInverse()
Category of rings

sage : _.Division()
Category of division rings

sage : _ & Sets().Finite()
Category of finite fields
```
Some more examples

sage : Mul = Magmas().Associative().Unital()
Category of monoids

sage : Add = AdditiveMagmas().AdditiveAssociative().AdditiveCommutative().AdditiveUnital()
Category of commutative additive monoids

sage : (Add & Mul).Distributive()
Category of semirings

sage : _.AdditiveInverse()
Category of rings

sage : _.Division()
Category of division rings

sage : _ & Sets().Finite()
Category of finite fields
Some more examples

```
sage: Mul = Magmas().Associative().Unital()
Category of monoids

sage: Add = AdditiveMagmas().AdditiveAssociative().AdditiveCommutative().AdditiveUnital()
Category of commutative additive monoids

sage: (Add & Mul).Distributive()
Category of semirings

sage: _.AdditiveInverse()
Category of rings

sage: _.Division()
Category of division rings

sage: _ & Sets().Finite()
Category of finite fields
```
Some more examples

```python
sage : Mul = Magmas().Associative().Unital()
Category of monoids

sage : Add = AdditiveMagmas().AdditiveAssociative().AdditiveCommutative().AdditiveUnital()
Category of commutative additive monoids

sage : (Add & Mul).Distributive()
Category of semirings

sage : _.AdditiveInverse()
Category of rings

sage : _.Division()
Category of division rings

sage : _ & Sets().Finite()
Category of finite fields
```
Full grown category

```python
@semantic(mmt = 'Semigroup')
class Semigroups(Category) :
    def super_categories() :
        return [Magmas()]

class ParentMethods : ...
    @abstract_method
    def semigroup_generators(self) :
        def cayley_graph(self) : ...

class ElementMethods : ...
    def __pow__(x, k) : ...

class MorphismMethods : ...

class CartesianProducts :
    def extra_super_categories(self) : return [Semigroups()]

class ParentMethods :
    def semigroup_generators(self) : ...

Unital = LazyImport('sage.categories.monoids', 'Monoids')
```
Implementation

Subposet of implemented categories

- Described by a spanning tree adding one axiom/construction at a time
- Size: $O($number of functions$)$

Fundamental operations

- joins, meets
- adding one axiom, applying one construction

Algorithmic

- Mutually recursive lattice algorithms
- Reasonable complexity ($\approx$ linear)
Implementation

Subposet of *implimented* categories

- Described by a spanning tree
  adding one axiom/construction at a time
- Size: $O(\text{number of functions})$

Fundamental operations

- joins, meets
- adding one axiom, applying one construction

Algorithmic

- Mutually recursive lattice algorithms
- Reasonable complexity ($\approx$ linear)
Implementation

Subposet of implemented categories

- Described by a spanning tree adding one axiom/construction at a time
- Size: $O(\text{number of functions})$

Fundamental operations

- joins, meets
- adding one axiom, applying one construction

Algorithmic

- Mutually recursive lattice algorithms
- Reasonable complexity ($\approx$ linear)
Summary

- **SageMath models a variety of mathematical objects**

- Supported by a large hierarchy of categories
  - Bookshelves for:
    - Semantic
    - Generic Code, Documentation, Tests
    - for parents, elements, morphisms, homsets
    - Axioms, Constructions, ...

- Robust: based on a century of abstract algebra

- Using Python’s standard Object Oriented features

- **Scaling**:
  - Dynamic construction of hierarchy of classes from the *semantic information* and *mixin classes* provided by the categories
  - Lattice algorithms
  - Control of the linearization for multiple inheritance (C3)
Summary

- **SageMath** models a variety of mathematical objects

- Supported by a **large** hierarchy of categories
  Bookshelves for:
  - Semantic
  - Generic Code, Documentation, **Tests**
  - for **parents**, **elements**, **morphisms**, **homsets**
  - Axioms, Constructions, ...

- Robust: based on a century of abstract algebra

- Using Python’s standard **Object Oriented** features

- **Scaling**:
  - Dynamic construction of hierarchy of classes from the *semantic information* and *mixin classes* provided by the categories
  - Lattice algorithms
  - Control of the linearization for multiple inheritance (C3)
Summary

- SageMath models a variety of mathematical objects

- Supported by a large hierarchy of categories
  Bookshelves for:
  - Semantic
  - Generic Code, Documentation, Tests
  - for parents, elements, morphisms, homsets
  - Axioms, Constructions, ...

- Robust: based on a century of abstract algebra

- Using Python’s standard Object Oriented features

- Scaling:
  - Dynamic construction of hierarchy of classes from the semantic information and mixin classes provided by the categories
  - Lattice algorithms
  - Control of the linearization for multiple inheritance (C3)
Summary

- SageMath models a variety of mathematical objects

- Supported by a large hierarchy of categories
  Bookshelves for:
  - Semantic
  - Generic Code, Documentation, Tests
  - for parents, elements, morphisms, homsets
  - Axioms, Constructions, ...

- Robust: based on a century of abstract algebra

- Using Python’s standard Object Oriented features

- Scaling:
  - Dynamic construction of hierarchy of classes from the semantic information and mixin classes provided by the categories
  - Lattice algorithms
  - Control of the linearization for multiple inheritance (C3)
Summary

- SageMath models a variety of mathematical objects

- Supported by a large hierarchy of categories
  Bookshelves for:
  - Semantic
  - Generic Code, Documentation, Tests
  - for parents, elements, morphisms, homsets
  - Axioms, Constructions, ...

- Robust: based on a century of abstract algebra

- Using Python’s standard Object Oriented features

- Scaling:
  - Dynamic construction of hierarchy of classes from the semantic information and mixin classes provided by the categories
  - Lattice algorithms
  - Control of the linearization for multiple inheritance (C3)
Additional benefits

Explicit representation of the knowledge

- Better formalization of the system
- Educational
- Easier to export

Tentative applications

- Math-in-the-Middle? Alignments?
- Automatic generation of interfaces between systems?
- Cross checking with other systems
- Documentation and navigation systems?
Additional benefits

Explicit representation of the knowledge

- Better formalization of the system
- Educational
- Easier to export

Tentative applications

- Math-in-the-Middle? Alignments?
- Automatic generation of interfaces between systems?
- Cross checking with other systems
- Documentation and navigation systems?
The paradigm is good; is this the right implementation?

Natural in its context

- A dynamical language (Python)
- Object oriented programming

Outside of this context?

Alternative implementations?

- In a language with static or gradual typing?
- Using templates or traits?
  For example in C++ or Scala
- Using multimethods
  For example in Julia or GAP
- In proof systems?
The paradigm is good; is this the right implementation?

Natural in its context

- A dynamical language (Python)
- Object oriented programming

Outside of this context?

Alternative implementations?

- In a language with static or gradual typing?
- Using templates or traits?
  - For example in C++ or Scala
- Using multimethods
  - For example in Julia or GAP
- In proof systems?
The paradigm is good; is this the right implementation?

Natural in its context

- A dynamical language (Python)
- Object oriented programming

Outside of this context?

Alternative implementations?

- In a language with static or gradual typing?
- Using templates or traits?
  For example in C++ or Scala
- Using multimethods
  For example in Julia or GAP
- In proof systems?
Collaborations welcome!

PhD or Postdoc grant 2016-2019!

Laboratoire de Recherche en Informatique
Université Paris Sud

(funded by OpenDreamKit)