Categories, axioms, constructions in SageMath: Modeling mathematics for fun and profit

Nicolas M. Thiéry

Laboratoire Interdisciplinaire des Sciences de l'information, Université Paris-Saclay

May 22nd of 2024
Trimester Program: "Prospects of Formal Mathematics"
Hausdorf Research Institute for Mathematics

Abstract

General purpose computational math systems such as `SageMath` systems provides thousands of mathematical objects and tens of thousands of operations to compute with them. We believe that 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 back in the early '10. 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 (e.g. GAP), and discuss open problems. This is meant as a basis for discussions: how are the equivalent challenges tackled in proof systems? Is there ground for cross-fertilization?

One challenge with general purpose math systems

Numbers: $42, \frac{7}{9}, \frac{I+sqrt(3)}{2}, \pi$, 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, ...

One challenge with general purpose math systems

Numbers: 42, $\frac{7}{9}$, $\frac{I+sqrt(3)}{2}$, π, 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, ...

One challenge with general purpose math systems

Numbers: 42, $\frac{7}{9}$, $\frac{I+sqrt(3)}{2}$, π, $2.71828182845904523536028747$?

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, ...

One challenge with general purpose math systems

Numbers: 42, $\frac{7}{9}$, $\frac{I+sqrt(3)}{2}$, π, $2.71828182845904523536028747?$

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, ...

One challenge with general purpose math systems

Numbers: $42, \frac{7}{9}, \frac{I+sqrt(3)}{2}, \pi, 2.718281828459045235360287 47?$

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, ...

One challenge with general purpose math systems

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

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, ...

One challenge with general purpose math systems

Numbers: 42, $\frac{7}{9}$, $\frac{I+sqrt(3)}{2}$, π, $2.718281828459045235360287\,47?$

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

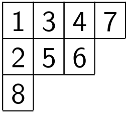
Polynomials: $-9x^8 + x^7 + x^6 - 13x^5 - x^3 - 3x^2 - 8x + 4$

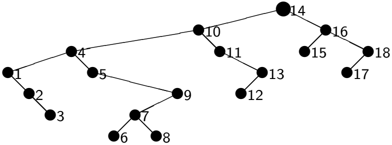
Series: $1 + 1x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \cdots$

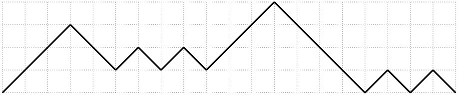
Symbolic expressions, equations: $cos(x)^2 + sin(x)^2 == 1$

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

Combinatorial objects

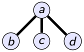




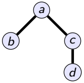


$$0100101001001010010100100101001001010010\cdots$$

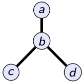
$$\frac{\frac{1}{6}q^2-\frac{1}{6}q}{q^5+2q^4+3q^3+3q^2+2q+1}$$



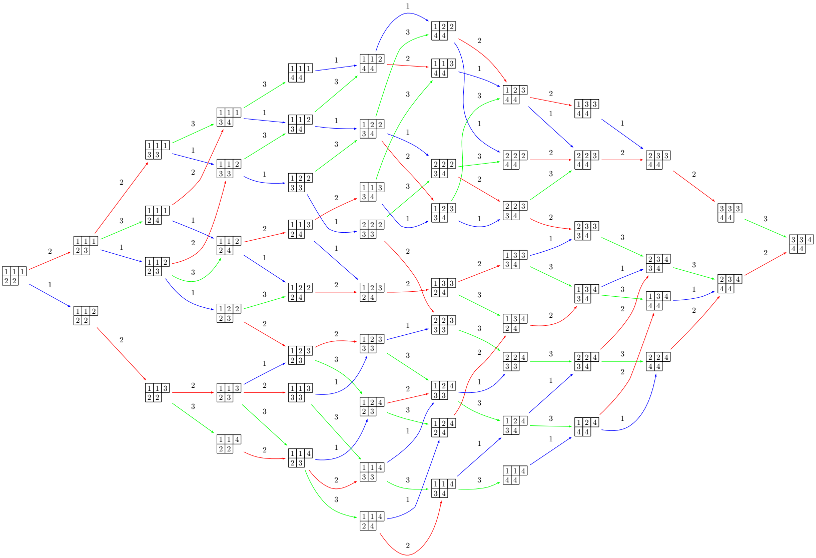
$$+\frac{q^2}{q^5+2q^4+3q^3+3q^2+2q+1}$$



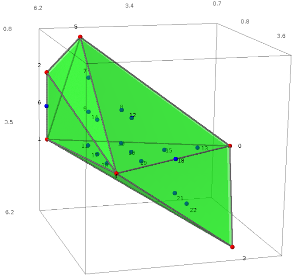
$$+\frac{\frac{1}{2}q}{q^4+q^3+2q^2+q+1}$$

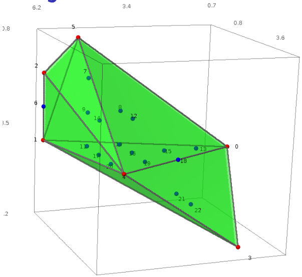


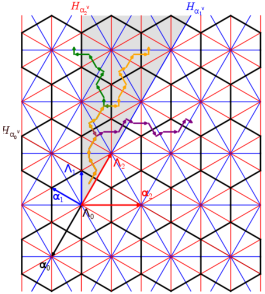
Graphs

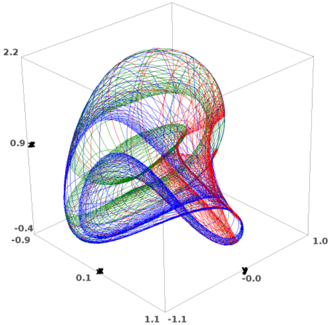


Geometric objects









Example: `SageMath`, a large library of mathematics objects and algorithms

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

Challenges

- How to structure this library?
- How to guide the user?
- How to promote consistency and robustness?
- How to reduce duplication?
- How to promote modularity?

Example: SageMath, a large library of mathematics objects and algorithms

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

Challenges

- How to structure this library?
- How to guide the user?
- How to promote consistency and robustness?
- How to reduce duplication?
- How to promote modularity?

Example: `SageMath`, a large library of mathematics objects and algorithms

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

Challenges

- How to structure this library?
- How to guide the user?
- How to promote consistency and robustness?
- How to reduce duplication?
- How to promote modularity?

What does typical math library look like?

Computational kernels

- Grinding integers, vectors, polynomials, permutations, Gröbner bases, permutations, combinatorial objects, ...
- All about data structures, memory management, parallelism orchestration, assembly optimization, ...

Recipes to reduce to the above

- Applying definitions and theorems!
- Recipe: assuming $C(x, y)$, $F(x, y)$ can be computed by:
  - A formula: $Foo(x) + Bar(y)$
  - Some change of representation: $FFT^{-1}(Foo(FFT(x)))$
  - Mapping Foo on all elements of x and reducing with Bar
  - ...
- A library of thousands of recipes
- A computation: recursive composition of dozens of them

Orchestrating recipes?

- for feature and expressiveness
- for performance

Some examples of orchestrators

- GAP's method selection
- Sage's method selection
  With inspiration from Axiom, MuPAD, OOP, ...
- Sage's coercion model
- GAP's CAP project: Categories, Algorithms, and Programming

Main approaches

- Method selection
- Morphism discovery

Orchestrating recipes?

- for feature and expressiveness
- for performance

Some examples of orchestrators

- GAP's method selection
- Sage's method selection
  With inspiration from Axiom, MuPAD, OOP, ...
- Sage's coercion model
- GAP's CAP project: Categories, Algorithms, and Programming

Main approaches

- Method selection
- Morphism discovery

Orchestrating recipes?

- for feature and expressiveness
- for performance

Some examples of orchestrators

- GAP's method selection
- Sage's method selection
  With inspiration from Axiom, MuPAD, OOP, ...
- Sage's coercion model
- GAP's CAP project: Categories, Algorithms, and Programming

Main approaches

- Method selection
- Morphism discovery

Example of recipe: 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 of recipe: 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 <u>generic</u> implementation!

Example: binary powering II

An algebraic realm

- Semigroup:
  a set S endowed with an associative binary internal law $*$

We want to

- Implement `pow_exp(x,k)`
- Provide a recipe
  - 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?
  Better recipes

Example: binary powering II

An algebraic realm

- Semigroup:
  a set S endowed with an associative binary internal law $*$

We want to

- Implement `pow_exp(x,k)`
- Provide a recipe
  - 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?
  Better recipes

Example: binary powering II

An algebraic realm

- Semigroup:
  a set S endowed with an associative binary internal law $*$

We want to

- Implement `pow_exp(x,k)`
- Provide a recipe
  - 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?
  Better recipes

Example: binary powering II

An algebraic realm

- Semigroup:
  a set S endowed with an associative binary internal law $*$

We want to

- Implement `pow_exp(x,k)`
- Provide a recipe
  - 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?
  Better recipes

Selection mechanism

We want

- Design a **hierarchy of realms** and specify the operations there
- Provide recipes for these 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 recipe for `f`

Selection mechanism

We want

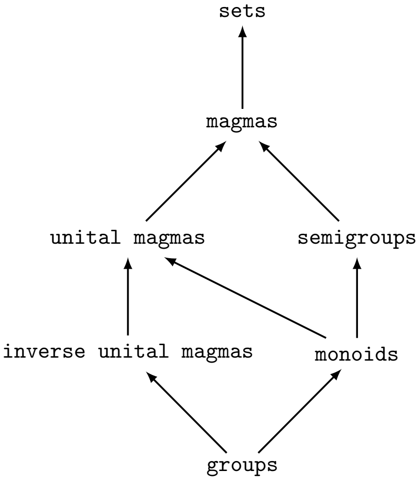
- Design a **hierarchy of realms** and specify the operations there
- Provide recipes for these operations
  Specify in which realm they are valid
- Specify in which realm each object is

We need a selection mechanism:

- to resolve the call `f(x)`
- by selecting the most specific recipe for `f`

Sage's hierarchy of realms (Categories)

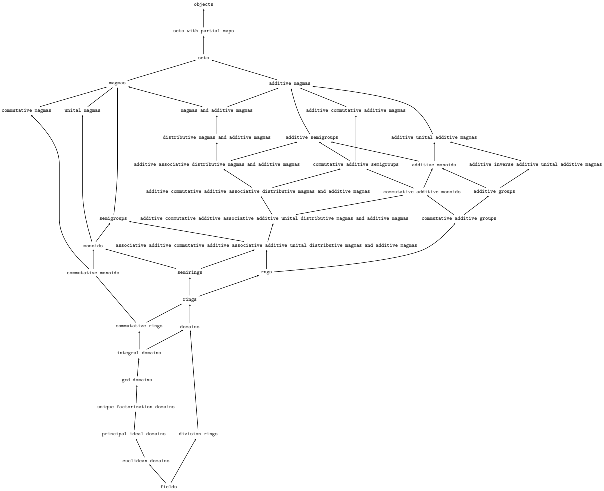
Model math concepts: Finite sets, Groups, Fields, Graphs, ...
By a **hierarchy of abstract classes**:



Sage's hierarchy of realms (Categories)

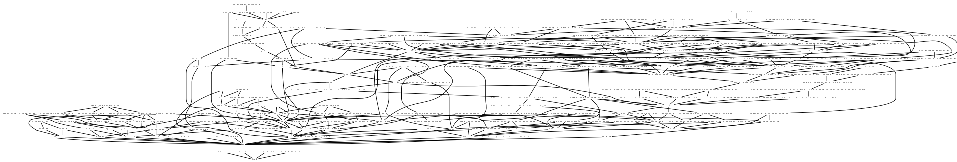
Model math concepts: Finite sets, Groups, Fields, Graphs, ...

By a **large hierarchy of abstract classes**:



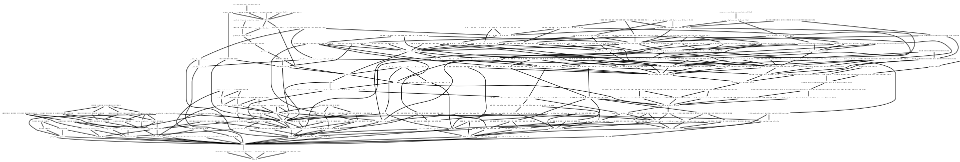
Sage's hierarchy of realms (Categories)

Model math concepts: Finite sets, Groups, Fields, Graphs, ...
By a **huge** **hierarchy of abstract classes**:



Sage's hierarchy of realms (Categories)

Model math concepts: Finite sets, Groups, Fields, Graphs, ...
By a **huge hierarchy of abstract classes**:



Totally insane???

Designing a hierarchy of realms

In general

- Hard problem: isolate the proper business concepts
- Recommendation: avoid large hierarchies
  use instead object composition to separate concerns

In mathematics

- "Few" fundamental concepts:
  - basic operations/structure: $\in$, $+$, *, cardinality, topology, ...
  - axioms: associative, finite, compact, ...
  - constructions: cartesian product, quotients, ...
- Concepts known by the end users
- All the richness comes from **combining** those few concepts to form many realms:
  groups, fields, semirings, lie algebras, ...

Designing a hierarchy of realms

In general

- Hard problem: isolate the proper business concepts
- Recommendation: avoid large hierarchies
  use instead object composition to separate concerns

In mathematics

- "Few" fundamental concepts:
  - basic operations/structure: $\in$, $+$, *, cardinality, topology, ...
  - axioms: associative, finite, compact, ...
  - constructions: cartesian product, quotients, ...
- Concepts known by the end users
- All the richness comes from **combining** those few concepts to form many realms:
  groups, fields, semirings, lie algebras, ...

Designing a hierarchy of realms

In general

- Hard problem: isolate the proper business concepts
- Recommendation: avoid large hierarchies
  use instead object composition to separate concerns

In mathematics

- "Few" fundamental concepts:
  - basic operations/structure: $\in$, $+$, *, cardinality, topology, ...
  - axioms: associative, finite, compact, ...
  - constructions: cartesian product, quotients, ...
- Concepts known by the end users
- All the richness comes from **combining** those few concepts to form many realms:
  groups, fields, semirings, lie algebras, ...

Designing a hierarchy of realms

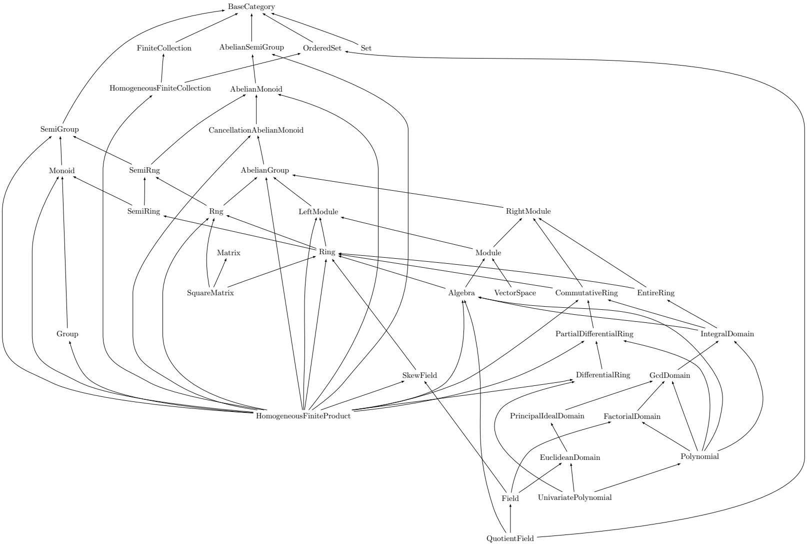
In general

- Hard problem: isolate the proper business concepts
- Recommendation: avoid large hierarchies
  use instead object composition to separate concerns

In mathematics

- "Few" fundamental concepts:
  - basic operations/structure: $\in$, $+$, *, cardinality, topology, ...
  - axioms: associative, finite, compact, ...
  - constructions: cartesian product, quotients, ...
- Concepts known by the end 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

Pioneers 1980- I

Axiom (?), Aldor(1990), MuPAD (1997)

- Bespoke 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 4, 1997

- Bespoke 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 x
- Implicit modeling of the hierarchy

Example

```
powExp := function(n, k) ...

InstallMethod(pow, [IsMagma, IsAssociative], powExp)
```

Related developments

Focal (Certified CAS)

- Species

MathComp (Proof assistant) 2013-(?)

- Canonical structures

MMT (Knowledge management) (?)

- E.g. LATIN's theories

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!

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

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

A concrete class

```
class MySemigroup(Semigroup):
    def semigroup_generators(self): ...
```

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?

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?

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: ...
```

A concrete class

```
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

```
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

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

Usage

```
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'>, ...]
```

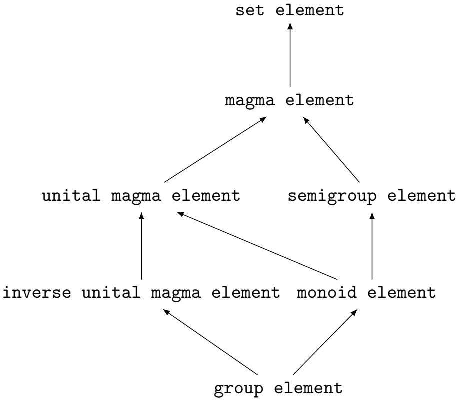
Generic tests

```
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:

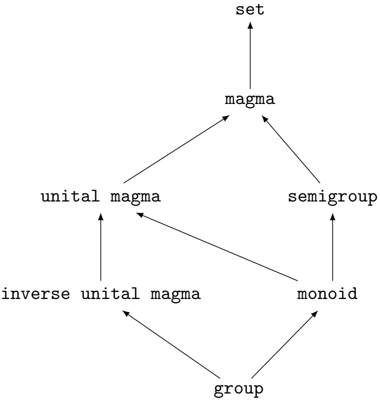


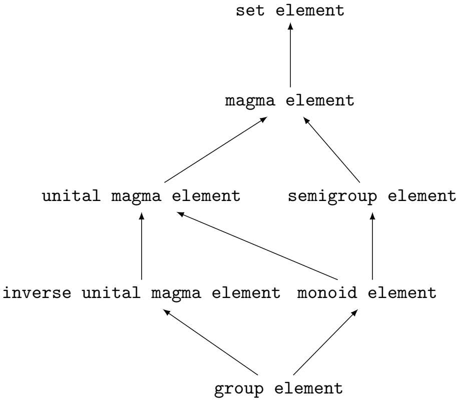
- the concrete classes for parents and elements

How does this work?

Dynamic construction, from the mixins, of:

- three hierarchies of abstract classes:



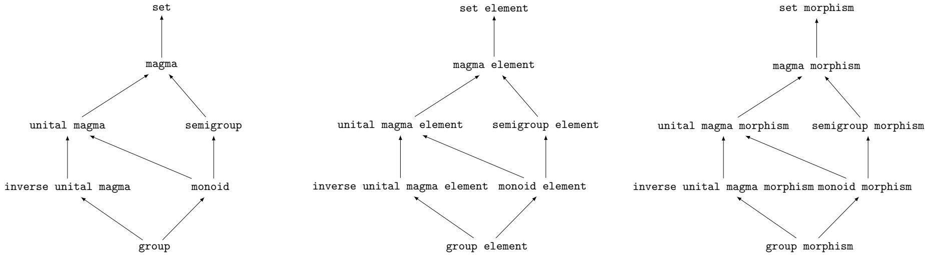


- the concrete classes for parents and elements

How does this work?

Dynamic construction, from the mixins, of:

- three hierarchies of abstract classes:



- 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 (and Interoperability?)

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 (and Interoperability?)

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 (and Interoperability?)

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 (and Interoperability?)

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

```
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 m
 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, 1.00000000000000, 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 m
 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, 1.00000000000000, 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 m
 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, 1.00000000000000, 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 m
 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
```

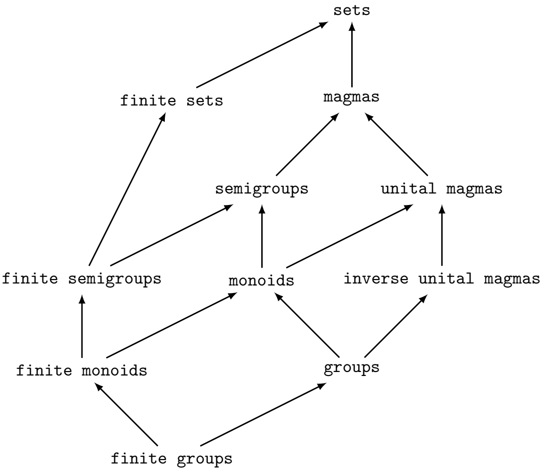
Taming the combinatorial explosion

Categories for groups:

$$
\begin{tikzcd}
 & \texttt{sets} & \\
 & \texttt{magmas} \arrow[u] & \\
\texttt{unital magmas} \arrow[ur] & & \texttt{semigroups} \arrow[ul] \\
\texttt{inverse unital magmas} \arrow[u] & & \texttt{monoids} \arrow[u] \arrow[ull] \\
 & \texttt{groups} \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

Taming the combinatorial explosion

Categories for finite groups:

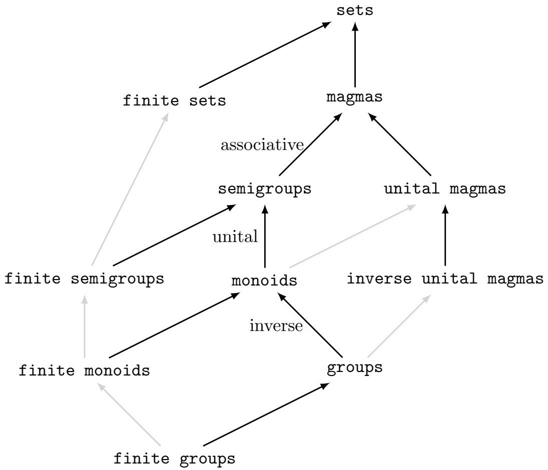


Implemented categories: 11 out of 14

Explicit inheritance: $1 + 9$ out of 15

Taming the combinatorial explosion

Categories for finite groups:

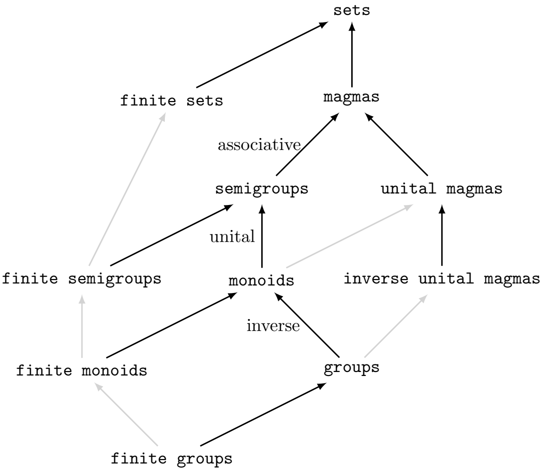


Implemented categories: 11 out of 14

Explicit inheritance: $1 + 9$ out of 15

Taming the combinatorial explosion

Categories for finite groups:

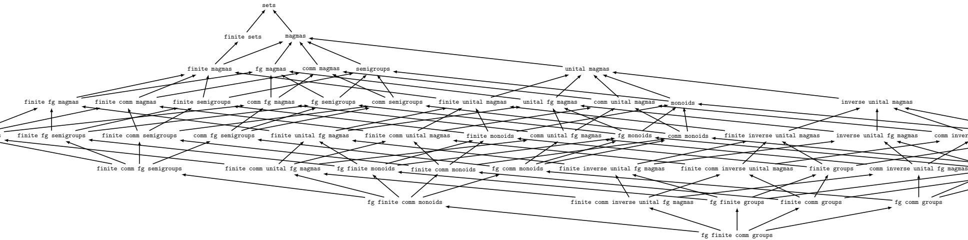


Implemented categories: 11 out of 14

Explicit inheritance: $1 + 9$ out of 15

Taming the combinatorial explosion

Categories for finitely generated finite commutative groups:

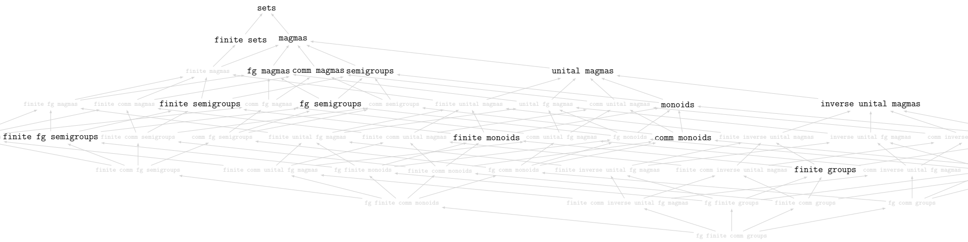


Implemented categories: 17 out of ≈ 54

Explicit inheritance: $1 + 15$ out of 32

Taming the combinatorial explosion

Categories for finitely generated finite commutative groups:

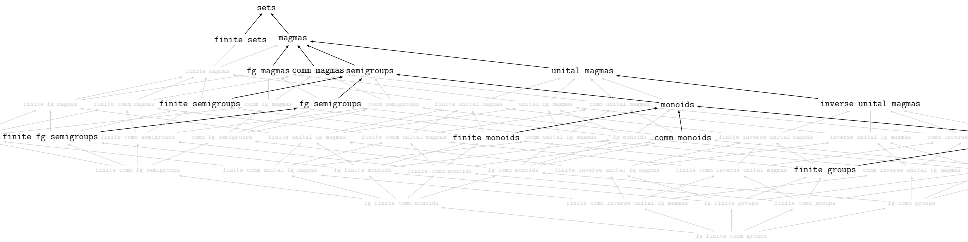


Implemented categories: 17 out of ≈ 54

Explicit inheritance: $1 + 15$ out of 32

Taming the combinatorial explosion

Categories for finitely generated finite commutative groups:

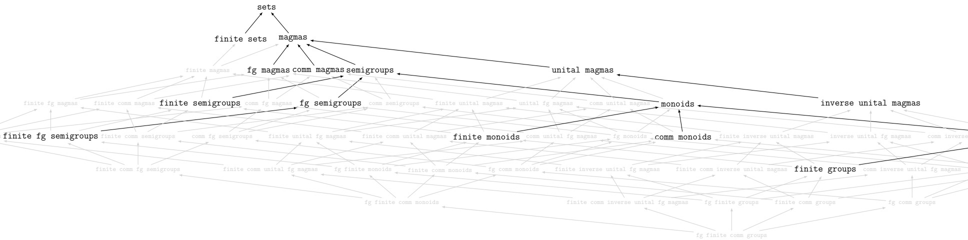


Implemented categories: 17 out of ≈ 54

Explicit inheritance: $1 + 15$ out of 32

Taming the combinatorial explosion

Categories for finitely generated finite commutative groups:

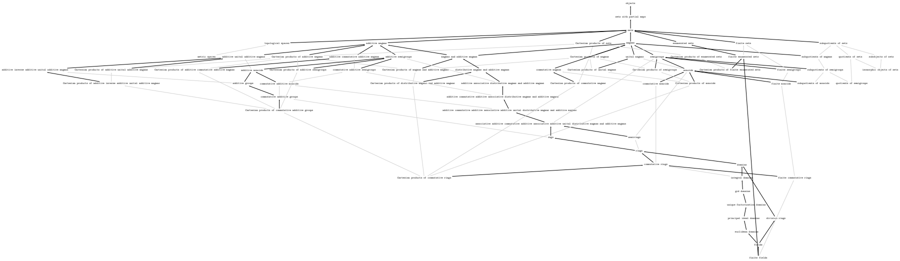


Implemented categories: 17 out of ≈ 54

Explicit inheritance: $1 + 15$ out of 32

Taming the combinatorial explosion

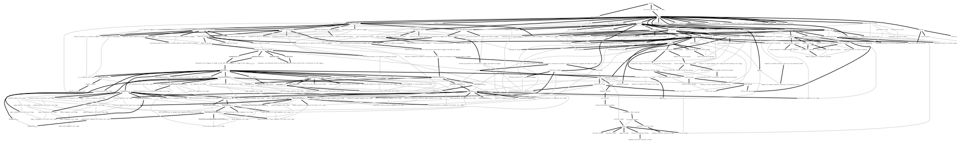
All implemented categories for fields:



Implemented categories: 71 out of $\approx 2^{13}$
Explicit inheritance: $3 + 64$ out of 121

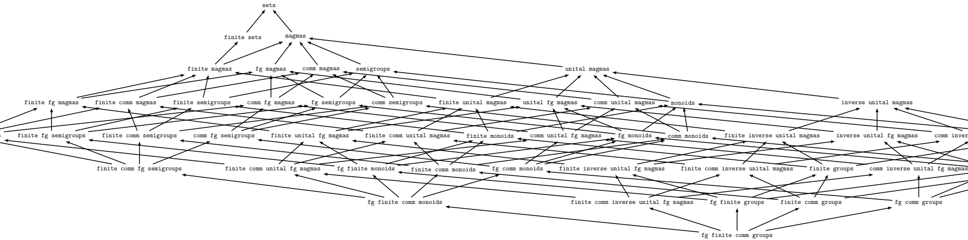
Taming the combinatorial explosion

All categories:



Categories: 265 out of $\approx 2^{50}$
Explicit inheritance: 70 out of 471

The hierarchy of categories as a lattice



- $\wedge$: objects in common

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

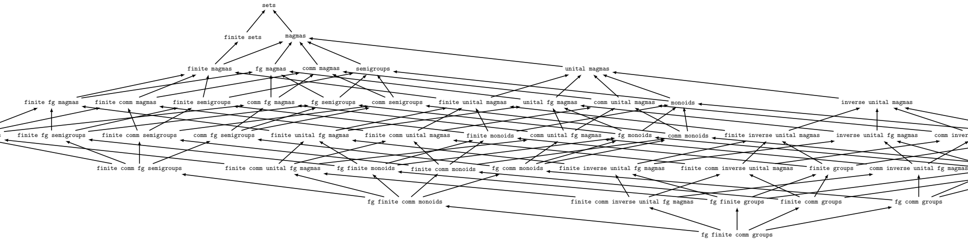
- $\vee$: structure in common

```
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 a lattice



- $\wedge$: objects in common

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

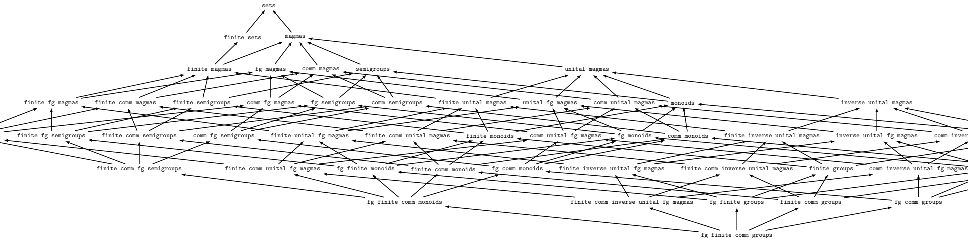
- $\vee$: structure in common

```
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



- $\wedge$: objects in common

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

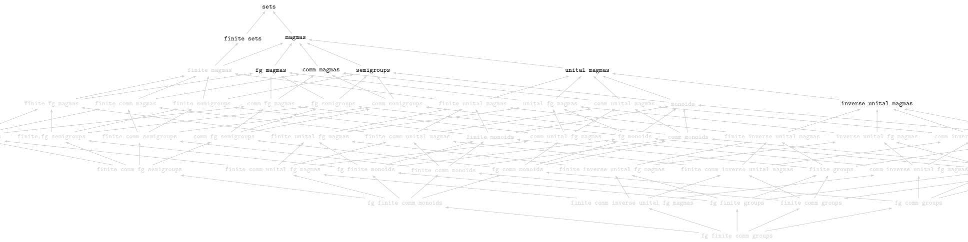
- $\vee$: structure in common

```
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



- $\wedge$: 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 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()
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: 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()
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: 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()
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: Magmas().Associative()  &  Magmas().Unital().Inverse()
Category of groups
```

Some more examples

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

```
sage: Add = AdditiveMagmas().AdditiveAssociative().AdditiveCommutati
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().AdditiveCommutati
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().AdditiveCommutati
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().AdditiveCommutativ
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().AdditiveCommutati
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().AdditiveCommutati
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

```
@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)

For fun and profit: formally proving the correctness?

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)

For fun and profit: formally proving the correctness?

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)

For fun and profit: formally proving the correctness?

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)

For fun and profit: formally proving the correctness?

Method Resolution Order (MRO)

How is multiple inheritance handled in Python?

class A	class B
class C(A,B)	

MRO: C, A, B

class B	class A
class D(B,A)	

MRO: D, B, A

Method Resolution Order computed by the **C3 algorithm**:

- Compatible with subclasses
- Compatible with the order of the bases
- Local

Now, what about:

class E(C, D)

Cannot create a consistent method resolution order (MRO)!

Method Resolution Order (MRO)

How is multiple inheritance handled in Python?

class A   class B

class C(A,B)

MRO: C, A, B

class B   class A

class D(B,A)

MRO: D, B, A

Method Resolution Order computed by the **C3 algorithm**:

- Compatible with subclasses
- Compatible with the order of the bases
- Local

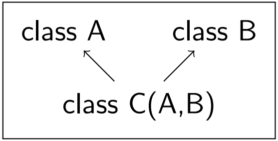
Now, what about:

class E(C, D)

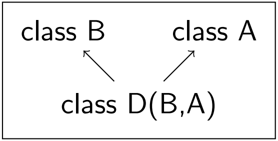
Cannot create a consistent method resolution order (MRO)!

Method Resolution Order (MRO)

How is multiple inheritance handled in Python?



MRO: C, A, B



MRO: D, B, A

Method Resolution Order computed by the **C3 algorithm**:

- Compatible with subclasses
- Compatible with the order of the bases
- Local

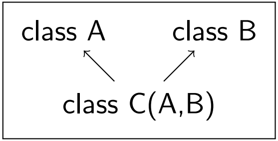
Now, what about:

class E(C, D)

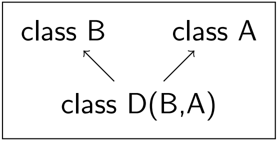
Cannot create a consistent method resolution order (MRO)!

Method Resolution Order (MRO)

How is multiple inheritance handled in Python?



MRO: C, A, B



MRO: D, B, A

Method Resolution Order computed by the **C3 algorithm**:

- Compatible with subclasses
- Compatible with the order of the bases
- Local

Now, what about:

class E(C, D)

Cannot create a consistent method resolution order (MRO)!

Method Resolution Order (MRO)

How is multiple inheritance handled in Python?

class A　　class B

class C(A,B)

MRO: C, A, B

class B　　class A

class D(B,A)

MRO: D, B, A

Method Resolution Order computed by the **C3 algorithm**:

- Compatible with subclasses
- Compatible with the order of the bases
- Local

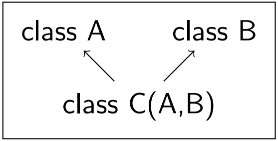
Now, what about:

class E(C, D)

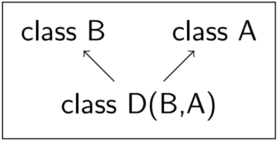
Cannot create a consistent method resolution order (MRO)!

Method Resolution Order (MRO)

How is multiple inheritance handled in Python?



MRO: C, A, B



MRO: D, B, A

Method Resolution Order computed by the **C3 algorithm**:

- Compatible with subclasses
- Compatible with the order of the bases
- Local

Now, what about:

$\boxed{\text{class E(C, D)}}$

Cannot create a consistent method resolution order (MRO)!

How to avoid MRO failures? Round 1

- **Choose** a global order on your classes
- Be consistent with it locally
- Failed!

  C3 does not know about your order:

MRO: E, D, B, A, C

How to avoid MRO failures? Round 1

- **Choose** a global order on your classes
- Be consistent with it locally
- Failed!

  C3 does not know about your order:

MRO: E, D, B, A, C

How to avoid MRO failures? Round 1

- **Choose** a global order on your classes
- Be consistent with it locally
- Failed!
  C3 does not know about your order:

$$
\begin{tikzcd}
\text{class B} & \text{class A} & \text{class C} \\
\text{class D(B,A)} \arrow[u] \arrow[ur] & & \\
\text{class E(D,C)} \arrow[u] \arrow[uurr] & &
\end{tikzcd}
$$

MRO: E, D, B, A, C

How to avoid MRO failures? Round 2

- **Find** some global order on your classes
- Be consistent with it locally
- Keeps failing over and over!

Math question: does there always **exist** some global order?
Answer: No!

How to avoid MRO failures? Round 2

- **Find** some global order on your classes
- Be consistent with it locally
- Keeps failing over and over!

Math question: does there always **exist** some global order?
Answer: No!

How to avoid MRO failures? Round 2

- **Find** some global order on your classes
- Be consistent with it locally
- Keeps failing over and over!

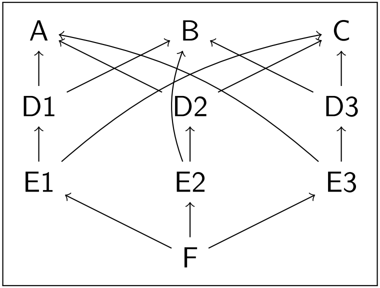
Math question: does there always **exist** some global order?

Answer: No!

How to avoid MRO failures? Round 2

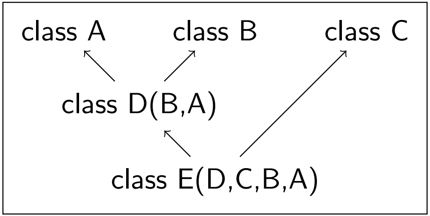
- **Find** some global order on your classes
- Be consistent with it locally
- Keeps failing over and over!

Math question: does there always **exist** some global order?
Answer: No!



How to avoid MRO failures? Round 3

- **Choose** your global order
- Force C3 to use **your** order:

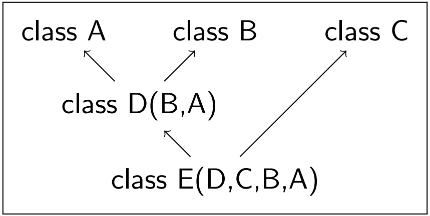


MRO: E, D, C, B, A

- Always works! Yeah!
- But:
  - Highly redundant: a maintenance nightmare!
  - Kills the algorithmic complexity of C3, dir, ...

How to avoid MRO failures? Round 3

- **Choose** your global order
- Force C3 to use **your** order:

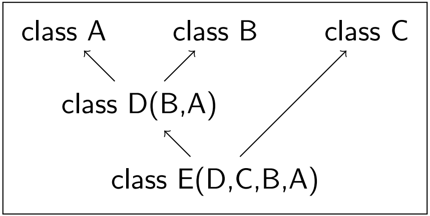


MRO: E, D, C, B, A

- Always works! Yeah!
- But:
  - Highly redundant: a maintenance nightmare!
  - Kills the algorithmic complexity of C3, dir, ...

How to avoid MRO failures? Round 3

- **Choose** your global order
- Force C3 to use **your** order:



MRO: E, D, C, B, A

- Always works! Yeah!

- But:
  - Highly redundant: a maintenance nightmare!
  - Kills the algorithmic complexity of C3, dir, ...

How to avoid MRO failures? Round 4

C3 under control:

- **Choose** your global order
- Run an instrumented version of C3
- Force the usual C3 to use **your** order

Et voilà!

- Always works
- Negligible overhead
- Fully automatic and transparent

`https://doc.sagemath.org/html/en/reference/misc/sage/misc/c3_controlled.html`

Hivert, T. 2022: Controlling the C3 super class linearization algorithm arXiv:2401.12740 [math.CO]

For fun and profit: formally proving the correctness?

How to avoid MRO failures? Round 4

C3 under control:

- **Choose** your global order
- Run an instrumented version of C3
- Force the usual C3 to use **your** order

Et voilà!

- Always works
- Negligible overhead
- Fully automatic and transparent

https://doc.sagemath.org/html/en/reference/misc/sage/misc/c3_controlled.html

Hivert, T. 2022: Controlling the C3 super class linearization algorithm arXiv:2401.12740 [math.CO]

For fun and profit: formally proving the correctness?

How to avoid MRO failures? Round 4

C3 under control:

- **Choose** your global order
- Run an instrumented version of C3
- Force the usual C3 to use **your** order

Et voilà!

- Always works
- Negligible overhead
- Fully automatic and transparent

`https://doc.sagemath.org/html/en/reference/misc/sage/misc/c3_controlled.html`

Hivert, T. 2022: Controlling the C3 super class linearization algorithm arXiv:2401.12740 [math.CO]

For fun and profit: formally proving the correctness?

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**
- **Twist for 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)
- Adoption?

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**
- **Twist for 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)
- Adoption?

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**
- **Twist for 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)
- Adoption?

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**
- **Twist for 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)
- Adoption?

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**
- **Twist for 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)
- Adoption?

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**
- **Twist for 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)
- Adoption?

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**
- **Twist for 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)
- Adoption?

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**
- **Twist for 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)
- Adoption?

Additional benefits

Explicit representation of the knowledge

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

Tentative applications

- Interoperability?
  DKKLPN 2016: Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach
  arXiv:1603.06424 [cs.MS]
- Automatic generation of interfaces between systems?
  Prototype: Sage-GAP interface
- Cross checking with other systems?
- Documentation and navigation systems?
  Prototype: Sage-Explorer

Additional benefits

Explicit representation of the knowledge

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

Tentative applications

- Interoperability?
  DKKLPN 2016: Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach
  arXiv:1603.06424 [cs.MS]
- Automatic generation of interfaces between systems?
  Prototype: Sage-GAP interface
- Cross checking with other systems?
- Documentation and navigation systems?
  Prototype: Sage-Explorer

The paradigm is good; what about the implementation?

Natural in its context

- A dynamical language (Python)
- Object oriented programming

Outside of this context?

- In a language with static or gradual typing?
- Using templates or traits?
  For example in C++ or Scala
- Using multimethods
  For example in **Julia** / **Oscar**
- In proof systems?
  Hierarchy builders are a growing thing, e.g. in Coq / MathComp

The paradigm is good; what about the implementation?

Natural in its context

- A dynamical language (Python)
- Object oriented programming

Outside of this context?

- In a language with static or gradual typing?
- Using templates or traits?
  For example in C++ or Scala
- Using multimethods
  For example in **Julia / Oscar**
- In proof systems?
  Hierarchy builders are a growing thing, e.g. in Coq / MathComp