REPORT ON OpenDreamKit DELIVERABLE D4.16

Exploratory support for semantic-aware interactive widgets providing views on objects represented and or in databases

ODILE BÉNASSY AND NICOLAS M. THIÉRY

Due on	31/08/2018 (M36)
Delivered on	31/08/2018
Lead	Université Paris-Sud (UPSud)

Progress on and finalization of this deliverable has been tracked publicly at:
`https://github.com/OpenDreamKit/OpenDreamKit/issues/90`

DELIVERABLE DESCRIPTION, AS TAKEN FROM GITHUB ISSUE #90 ON 2018-09-01

- **WP4:** User Interfaces
- **Lead Institution:** Université Paris-Sud
- **Due:** 2018-08-31 (month 36)
- **Nature:** Demonstrator
- **Task:** T4.5 (#73)
- **Proposal**: p. 48
- **Final report**

The Jupyter Notebook is a web application that enables the creation and sharing of executable documents containing live code, equations, visualizations, and explanatory text. Key features include Jupyter widgets which enable interactive visualization and can be composed to create full-featured interactive applications.

In this report, we explore the potential of Jupyter widgets for (pure) mathematics. The unique challenge comes from the huge variety of mathematical objects that the user may want to visualize and interact with, and the variety of graphical representations.

Over the course of four PM's, we pursued two main directions: the interactive visualization and edition of combinatorial objects (sage-combinat-widgets) and the interactive exploration of Sage features based on a display of mathematical objects enriched with contextual semantic information (sage-combinat-explorer).

Contents

1. Introduction

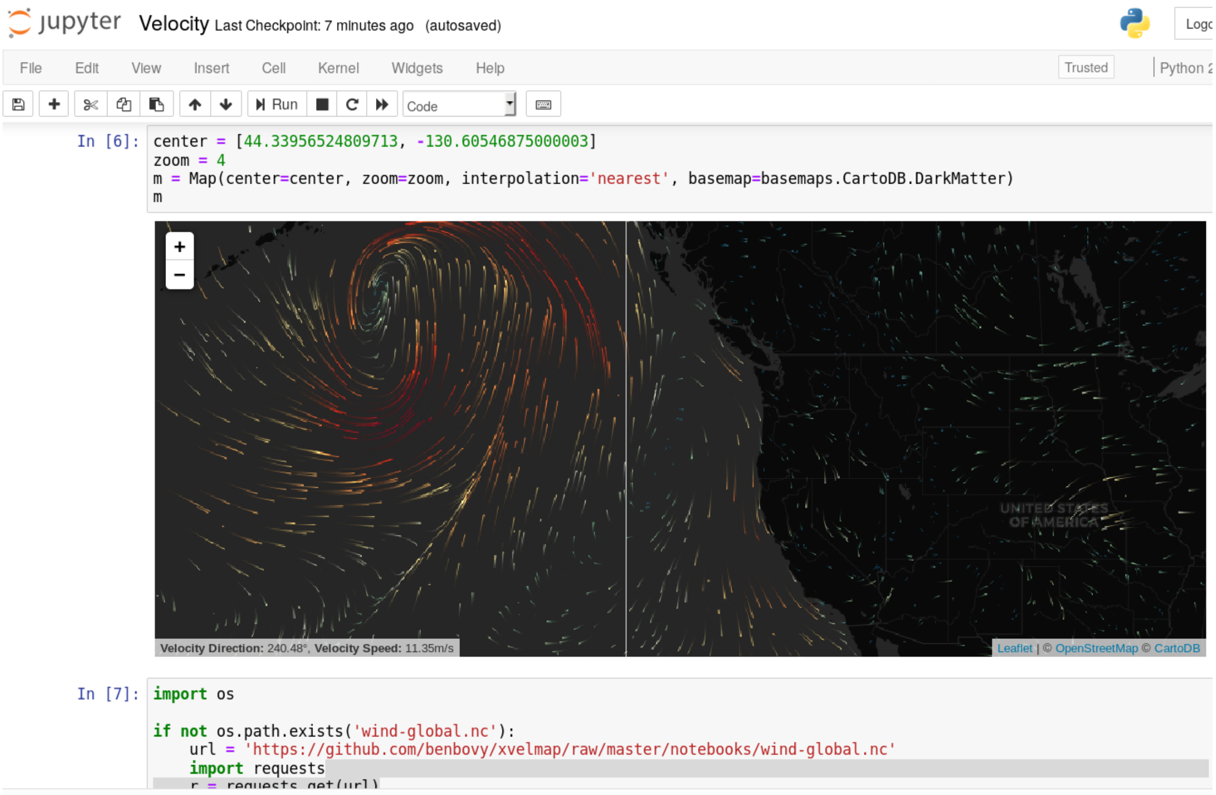
The Jupyter Notebook is a web application that enables the creation and sharing of executable documents containing live code, equations, visualizations and explanatory text. Reaching far beyond the standard REPL interaction (Read-Eval-Print Loop), a key feature of Jupyter is its Interactive widgets which enable real time interactive data visualizations; the Jupyter community has developed a large array of widgets for interactive 2D and 3D visualization of data in the form of charts, maps, tables, etc; See Figure 1 for an example, and D4.12 for ODK's contribution to 3D visualization. Furthermore, widgets can be *composed* to build rich applications, with all the usual UI components (e.g. menus, sliders, or layout control); see Figure 2. Hence, the Jupyter stack provides a very flexible environment catering for use cases ranging from a novice user typing just a few commands or browsing interactive documents to more advanced users authoring rich interactive applications for their fellows.

The question we are tackling in this report is how this technology – and specifically Jupyter widgets – can be leveraged for pure mathematics. The unique challenge comes from the huge variety of mathematical objects that the user may want to visualize and interact with, and the variety of graphical representations; see Figure 3 for some examples. We therefore can't hope to provide hand crafted solutions for each situation; instead we need to devise a toolbox of generic solutions from which users can easily derive specialized visualizations for their own pet objects.

We pursue two directions:

In Section 2, we explore the development of widgets for the graphical visualization in combinatorics – or more generally discrete mathematics. The choice of this area of mathematics was, to some extent out of personal interest and expertise, but more importantly because devising good representations – mental images – of discrete objects is at the heart of research in combinatorics. We start by reviewing in Section 2.1 the GAP package Francy which currently explores the natural use case of objects that admit a graph-like representation: trees, graphs, lattices of subgroups, crystals, posets, discrete markov chains, just to name a few. At this stage, ODK has a light contribution to the development of this package, through the supervision of OpenDreamKit's member Markus Pfeiffer. We highlight the lessons learned there and describe upcoming collaboration toward bringing Francyś features to SageMath.

Then, in Section 2.3, we report on the implementation in SageMath of a generic widget for objects that admit a representation as a collection of cells on a 2D grid, and specializations for typical objects such as partitions, tableaux, polyominos, aztec diamonds, mazes. One aspect which we explore beyond Francy is not only *interactive visualization* but also *interactive edition*,

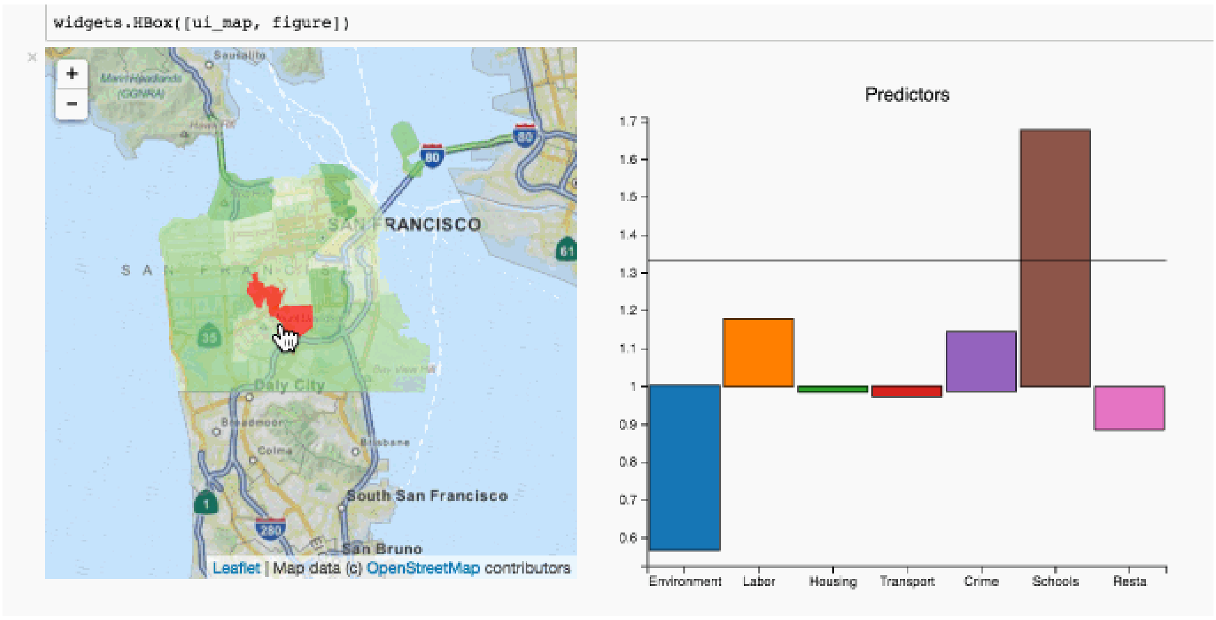

jupyter
Velocity Last Checkpoint: 7 minutes ago (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 2
Run
Code
Velocity Direction: 240.48°, Velocity Speed: 11.35m/s
Leaflet | © OpenStreetMap © CartoDB
UNITED STATES OF AMERICA

```
In [6]: center = [44.33956524809713, -130.60546875000003]
        zoom = 4
        m = Map(center=center, zoom=zoom, interpolation='nearest', basemap=basemaps.CartoDB.DarkMatter)
        m
```

```
In [7]: import os

        if not os.path.exists('wind-global.nc'):
            url = 'https://github.com/benbovy/xvelmap/raw/master/notebooks/wind-global.nc'
            import requests
```

FIGURE 1. A Jupyter widget displaying an interactive map based on OpenStreetMap data, overlayed with a visualisation of wind velocity data (courtesy of the ipyleaflet documentation).



```
widgets.HBox([ui_map, figure])
```

FIGURE 2. A Jupyter application displaying survey results on San Francisco city districts (courtesy of the ipyleaflet documentation).

where the underlying mathematical object gets changed by the user gestures. This is the occasion to explore the flexibility of the the design patterns emerging from Francy.

The choice of this use case was motivated by two important features:

- the visualization part is relatively simple making it a good case for prototyping;

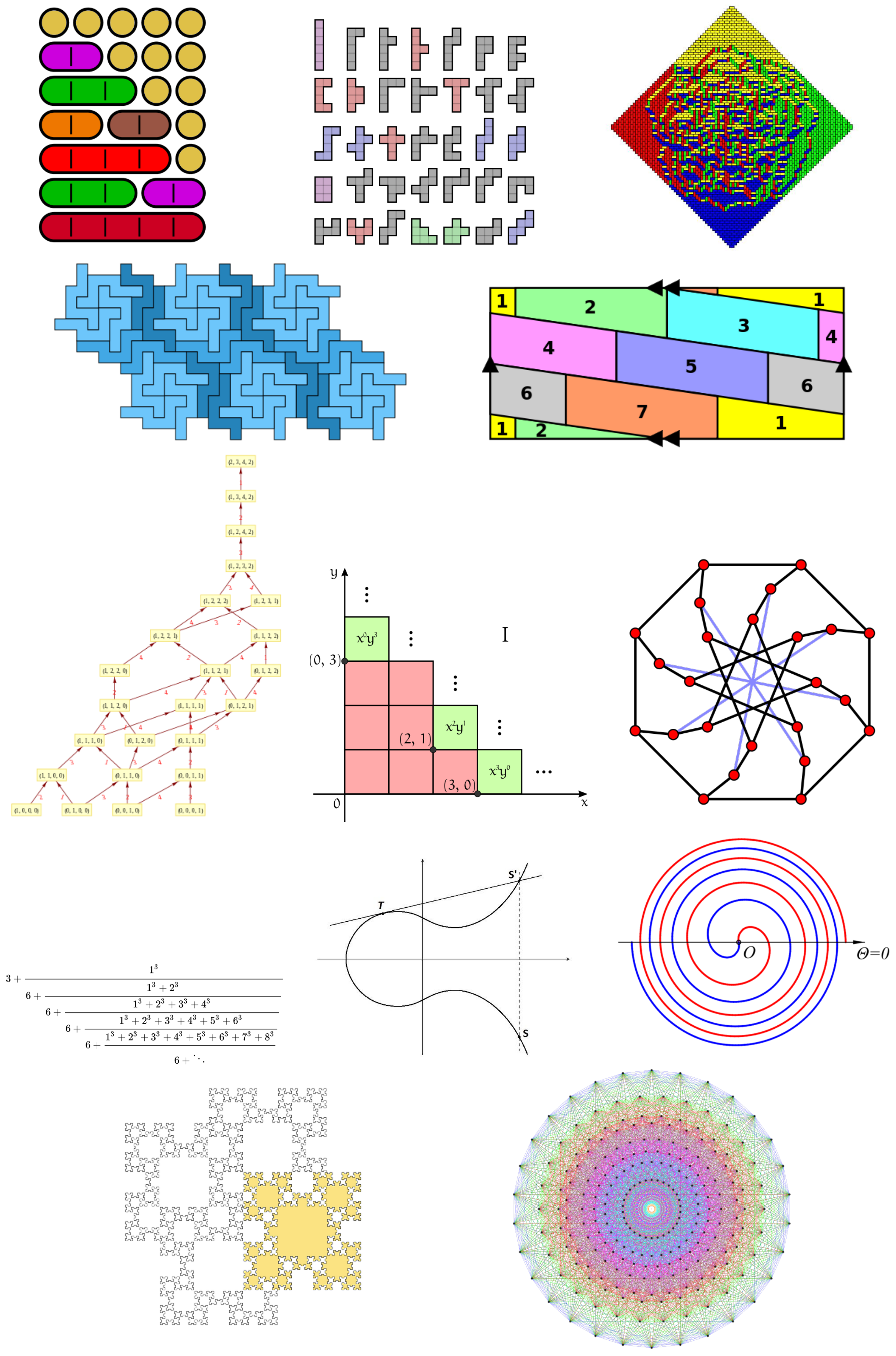
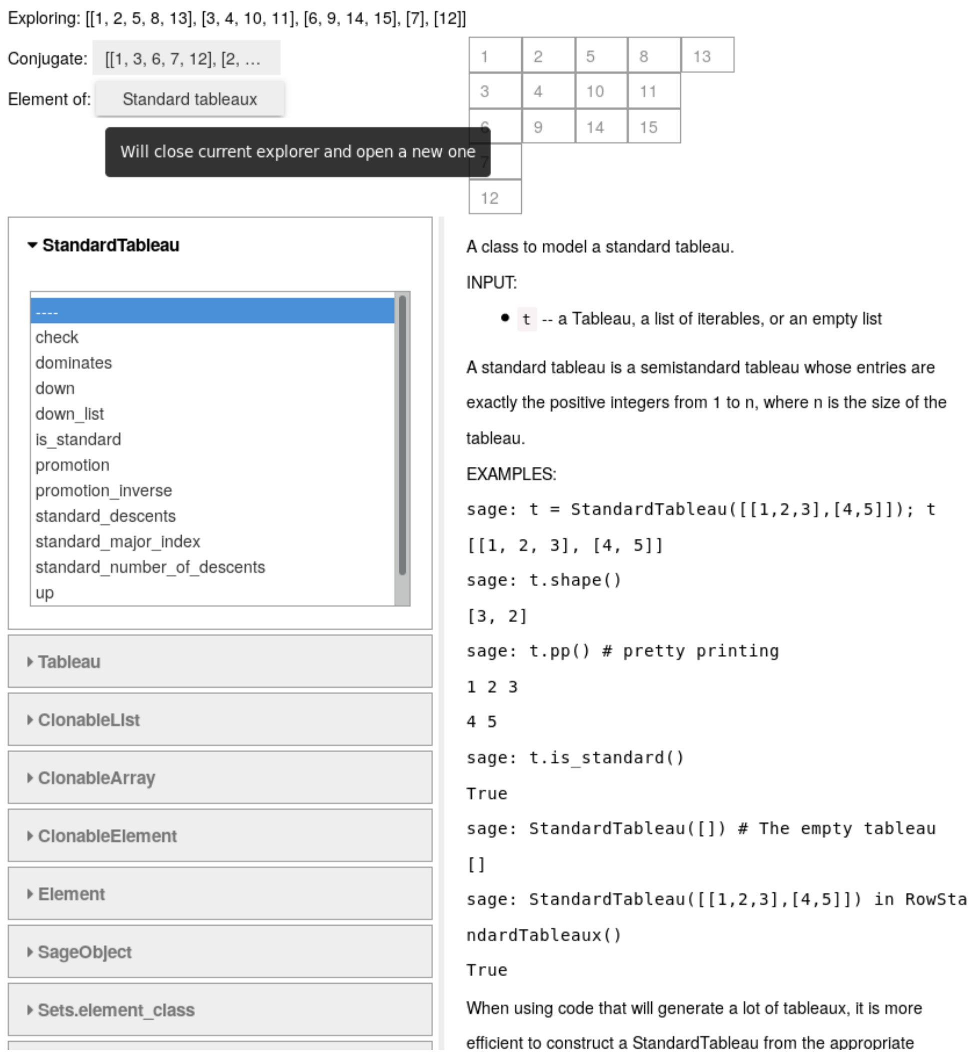


FIGURE 3. Some graphical visualizations of mathematical objects

- the potential impact is high due to the large number of objects covered, bringing in opportunities for testing how users manage to to adapt our generic tools for their own pet objects;

In a second direction, we explore in Section 3 the use of Jupyter widgets not only for the graphical visualization of an object, but for displaying a page offering a synthetic overview of the information about that object, including type, important properties and invariants, available operations, related objects, documentation, etc. All sort of information that is readily available by introspection but that a UI can make easy to discover and emphasize according to relevance. The challenge is that, given the huge variety of objects in a system like SageMath, we can't afford to hand craft such pages for each type of objects. We report on our prototype Sage-Explore that exploits the semantic embedded into the system to produce a reasonable overview page automatically tailored to each object.



Exploring: [[1, 2, 5, 8, 13], [3, 4, 10, 11], [6, 9, 14, 15], [7], [12]]

Conjugate: [[1, 3, 6, 7, 12], [2, ...

Element of: Standard tableaux

Will close current explorer and open a new one

1	2	5	8	13
3	4	10	11	
6	9	14	15	
7				
12				

▾ StandardTableau

----
check
dominates
down
down_list
is_standard
promotion
promotion_inverse
standard_descents
standard_major_index
standard_number_of_descents
up

▸ Tableau

▸ ClonableList

▸ ClonableArray

▸ ClonableElement

▸ Element

▸ SageObject

▸ Sets.element_class

A class to model a standard tableau.

INPUT:

- `t` -- a Tableau, a list of iterables, or an empty list

A standard tableau is a semistandard tableau whose entries are exactly the positive integers from 1 to n, where n is the size of the tableau.

EXAMPLES:

```
sage: t = StandardTableau([[1,2,3],[4,5]]); t
[[1, 2, 3], [4, 5]]
sage: t.shape()
[3, 2]
sage: t.pp() # pretty printing
1 2 3
4 5
sage: t.is_standard()
True
sage: StandardTableau([]) # The empty tableau
[]
sage: StandardTableau([[1,2,3],[4,5]]) in RowSta
ndardTableaux()
True
```

When using code that will generate a lot of tableaux, it is more efficient to construct a StandardTableau from the appropriate

FIGURE 4. Interactive introspection on a standard tableau with Sage-Explorer

Altogether, the Jupyter Widget technology has proven as mature and flexible as we hoped for. Designing new widgets does take some expertise, but from the experience we gained, we are confident that it lends itself well to the implementation of generic widgets by power users that can be specialized by casual users and used by novices.

Three main challenges arise along the way which we will continue to explore and tackle during the last year of OpenDreamKit:

- Implementing visualization widgets is time consuming and requires specific expertise that developers of computer algebra systems usually don't have. It's therefore desirable to reuse them across mathematical objects that share a certain type of visualization, and across computer algebra systems. How to best design widgets to decouple them from the specific objects or programming language at hand?
- User interface and visualization technologies tend to evolve much faster than computational systems; how to best design widgets to decouple the semantic part of the visualization of the widgets from the actual visualization technology, in order to prepare for a smooth and cheap migration path when such technology evolves and foster long term sustainability?
- Widgets, as all modern web technologies, rely a lot on asynchronous execution. This is quite a different model from the Read-Eval-Print main loop that many not-so-young computational systems (including SageMath!) have originally been designed for, and such systems don't always behave well under asynchronous pressure: we have faced some bad crashes. How deep is the difficulty? How hard will it be to resolve it?

2. INTERACTIVE VISUALIZATION OF COMBINATORIAL OBJECTS

2.1. Francy: an Interactive Discrete Mathematics Framework for GAP

Francy is a package for interactive visualisation in discrete mathematics. It's developed mainly by Manuel Martins, with supervision by ODK's member Markus Pfeiffer. For now, Francy has been focusing on the interactive visualisation of objects that admit a graph-like representation (trees, graphs, ...). Features include for example zoom-and-pan, moving nodes, clustering, menus for activating/deactivating interactions, etc. Those are basic interactive graph drawing features and are not particularly impressive by themselves: after all, interactive graph drawing and editing has been the subject of a whole community, with series of conferences such as the *International Symposium on Graph Drawing and Network Visualization*, and acclaimed tools like graphviz or tulip . What makes Francy stand is to bring those feature to computational systems and, more importantly, explore how to best leverage the technology to empower users of such systems to adapt it to their needs.

At this stage, Francy is a package for the GAP computer algebra system. However its features would be extremely valuable for other systems and in particular SAGE. Also Francy uses Jupyter Widgets as GUI framework and D3JS javascript library library for visualisation. While those are flourishing technologies that are promising for the years to come, lessons learned the hard way from the past (e.g. within the xgap project) shows that, at the scale of decades, GUI and visualization frameworks change at a faster pace than computational systems and visualization needs.

To tackle these two tensions, Francy has taken the approach of decoupling itself from the computational system and from the GUI framework and visualization libraries. It does this by introducing an intermediate layer of semantic models of what graphical representations are. With this approach, integrating a new computational system reduces to providing a backend describing the desired graphical representations for its mathematical objects in the semantic model. Similarly, integrating GUI framework reduces to providing a frontend describing how to concretely render and interact with the graphical representations from the model. In practice, the semantic model takes the form of an intermediate JSON data format, with JSON data being passed from the computational system to the browser.

2.2. Toward interactive edition

We have seen a *design pattern* emerging in the previous section: without this being formalized, a similar approach had been taken in Sage for static graph drawing, using the so-called "dot"

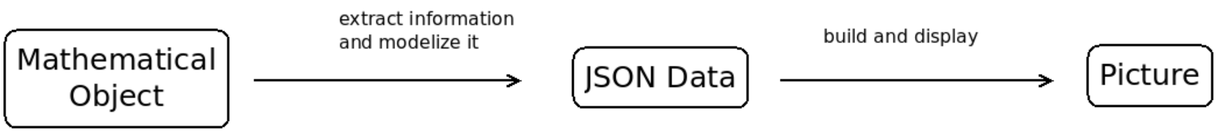


FIGURE 5. Interactive visualization Data Flow

format of the graphviz project as intermediate data format, to later produce graph pictures in a variety of formats (latex, pdf, svg).

Similarly, in the MuPAD-Combinat project a generic intermediate data format was designed for all objects that can be drawn by filling out certain cells in a 2-dimensional grid. For example Figure 6 shows a drawing of *skew tableau*, an important kind of combinatorial object arising in representation theory. We will call such a drawing a *grid-like representation*.

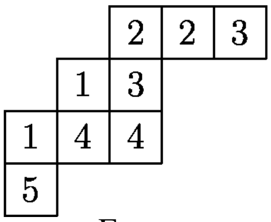


FIGURE 6. A skew-tableau

From the intermediate data format, drawings could be produced in a variety of format: latex, pdf, various forms of ascii art, . . . ; see Combinatorial Class With 2DBoxedRepresentation. This achieved the same two levels of decoupling as we have seen in Francy.

We decided to build on this previous experience to explore not only visualization, but also *interactive editing*: here, the interaction not only affects the visualization, but also induces changes to the the original mathematical objects via the intermediate data format. Thus the data flow is now bidirectional.

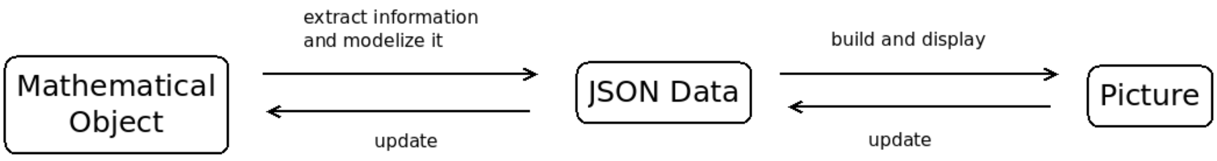


FIGURE 7. Interactive edition Data Flow

This brings new challenges.

First, updates may fail: given that the intermediate data format does not carry all the semantic of the mathematical objects, some interactive editions may be invalid; yet such editions should not be necessarily be forbidden, as it may take several elementary edition step to move from one valid state to another (think of a text editor that would block typing any key leading to a temporarily incorrectly spelled word!). Hence some logic is needed to handle loss and recovery of sync between the state of the semantic representation model and that of the underlying mathematical object.

In addition, rich editing features, requires richer message passing than just sending description of the object back and forth. For example, in order to provide visual hints to the user, the view may need to request what are the next valid edition steps for a given object; or in case an invalid step is taken, the object may return rich error messages enabling visual feedback. Also, for editing large objects, one may wish as well to send only incremental updates.

Therefore, the semantic representation model needs not only to specify a *data format* to represent instances of this model, but also *operations* on them, that is an API.

Aiming for interactive editing was suggested by the Larch Environment. In his master thesis (2013), Geoffrey W. French presents his own graphical interactive programming environment called *larch*. One of his main ideas is to *coerce* objects into graphical representations, meaning that every object must know how it can be graphically represented. Moreover, this representation is only a default one and can be tailored by the user: G.W. French speaks of different *perspectives* for the same object. The Larch Environment also maintained a state of objects in order to automatically refresh the representation. This project was meant as an experimental platform, without a strong and widely used underlying infrastructure as JUPYTER, and was discontinued in 2014. Nevertheless the online videos remain inspiring.

2.3. **A generic widget for objects with a grid-like representation**

To explore interactive editing, we set as goal the implementation of a generic Jupyter widget meant for all SAGE-objects admitting a grid-like representation.

This use case combined several advantages:

- The UI part was relatively straightforward, for example requiring no Javascript side extension;
- Little mathematical background was required; this, together with the previous point, made for a smooth learning curve for our Research Software Engineer;
- It was a low hanging fruit with a large coverage in algebraic combinatorics, including matrices, integer partitions, (standard) (skew) tableaux, ribbons, ribbon tableaux, grid graphs, aztec diagrams, etc.
- The end result is immediately useful to colleagues, enabling early feedback from users;
- There is a large variety of potential specializations, each with it's own quirks and specifics.

Hence, this use case provided a unique challenge: all the difficulty resided in the the design of a generic editing solution that encapsulates as much of the technicalities as possible, enabling users to specialize the generic solution for their own pet objects with little expertise.

2.4. **Overview of the features and design**

The developed JUPYTER widget *GridViewWidget* has the following features:

- editing the content of cells
- adding or removing cells
- selecting cells for more interactions
- supporting temporarily invalid states
- recovering the underlying mathematical object for further computations.

The semantic representation model takes the form of a Python class *GridViewEditor* that is both agnostic of the GUI and of the mathematical objects. To enable the latter, the interaction with mathematical objects of a certain kind, say integer partitions, goes through an adapter class which implements the required API on top of that of integer partitions. See the Figure 9.

The implementation of such an adapter shall be as simple as possible, to empower users to write new ones for their own pet mathematical objects. In particular, it should only require knowledge about the business logic of ditto mathematical objects. We explored different options, and battlefield tested them by writing adapters for a selection of mathematical objects of different nature, chosen in collaboration with potential users at the occasion of the experimental mathematics Spring school MathExp 2018 coorganised by OpenDreamKit (see D2.11: "Community building: Impact of development workshops, dissemination and training activities, year 2 and 3"). The final design will be chosen according to users feedback.

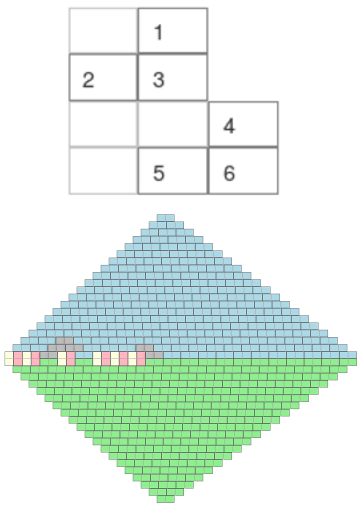


FIGURE 8. Editing a skew tableau, and flipping dominos on a grid graph

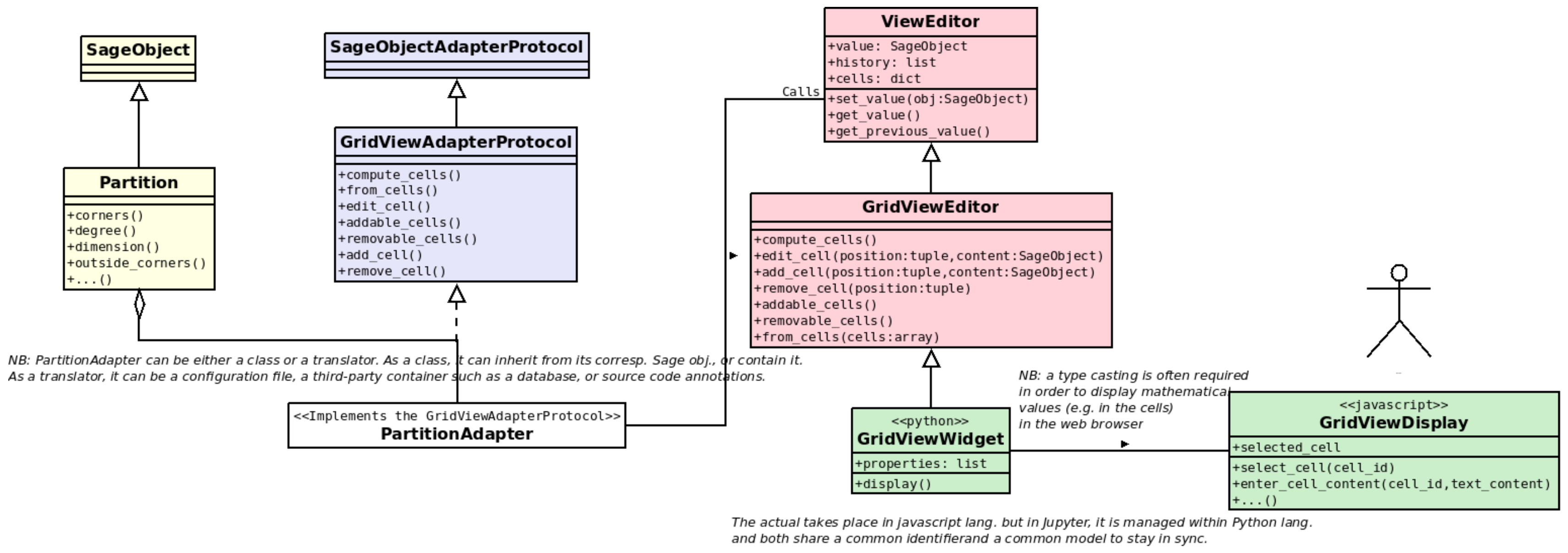

SageObject
SageObjectAdapterProtocol
ViewEditor
+value: SageObject
+history: list
+cells: dict
+set_value(obj:SageObject)
+get_value()
+get_previous_value()
Calls
Partition
+corners()
+degree()
+dimension()
+outside_corners()
+...()
GridViewAdapterProtocol
+compute_cells()
+from_cells()
+edit_cell()
+addable_cells()
+removable_cells()
+add_cell()
+remove_cell()
GridViewEditor
+compute_cells()
+edit_cell(position:tuple,content:SageObject)
+add_cell(position:tuple,content:SageObject)
+remove_cell(position:tuple)
+addable_cells()
+removable_cells()
+from_cells(cells:array)
NB: PartitionAdapter can be either a class or a translator. As a class, it can inherit from its corresp. Sage obj., or contain it.
As a translator, it can be a configuration file, a third-party container such as a database, or source code annotations.
<<Implements the GridViewAdapterProtocol>>
PartitionAdapter
NB: a type casting is often required
in order to display mathematical
values (e.g. in the cells)
in the web browser
<<python>>
GridViewWidget
+properties: list
+display()
<<javascript>>
GridViewDisplay
+selected_cell
+select_cell(cell_id)
+enter_cell_content(cell_id,text_content)
+...()
The actual takes place in javascript lang. but in Jupyter, it is managed within Python lang.
and both share a common identifierand a common model to stay in sync.

FIGURE 9. Sage View Editor classes

2.5. Dissemination strategy

The code is distributed as a Sage package sage-combinat-widgets, endowed with a Binder-based online demo. This package is meant to grow beyond this initial seed, attracting contributions from the community, and presumably be integrated progressively into SageMath.

The following steps will be taken to attract users and contributions:

- Implementing adapters for the most classical object types, and only them;
- Writing tutorials on how to write new ones;
- Running tutorials at the occasion of local work groups at *Laboratoire de Recherche Informatique* and *Laboratoire d'Informatique de l'École Polytechnique*, and at upcoming Sage Days.
- Inviting users to provide feedback, either face to face, on the development mailing list, or on the project *sage-combinat-widgets* issues interface.

2.6. Future plans

In the upcoming year, we plan to collaborate tightly with Francy to:

- Bring Francy's features to SAGE by implementing a backend. This would finally bring a solution to a long time yet pressing need of the community that was so far only partially fulfilled by unsustainable prototypes like Sage's graph editor.
- Further explore the semantic representation model design pattern with over kinds of graphical representations, with a similar decoupling between a SAGE backend, a semantic representation model, and a Jupyter/JS frontend.
- Contribute the models and frontends to Francy, making the features available to GAP.
- Seek collaboration with Macaulay2's Visualize package that share similar aims (though not yet Jupyter based).
- Write tutorials and collect best practices on how to develop new graphical representations on top of Jupyter widgets.

3. SAGE-EXPLORER

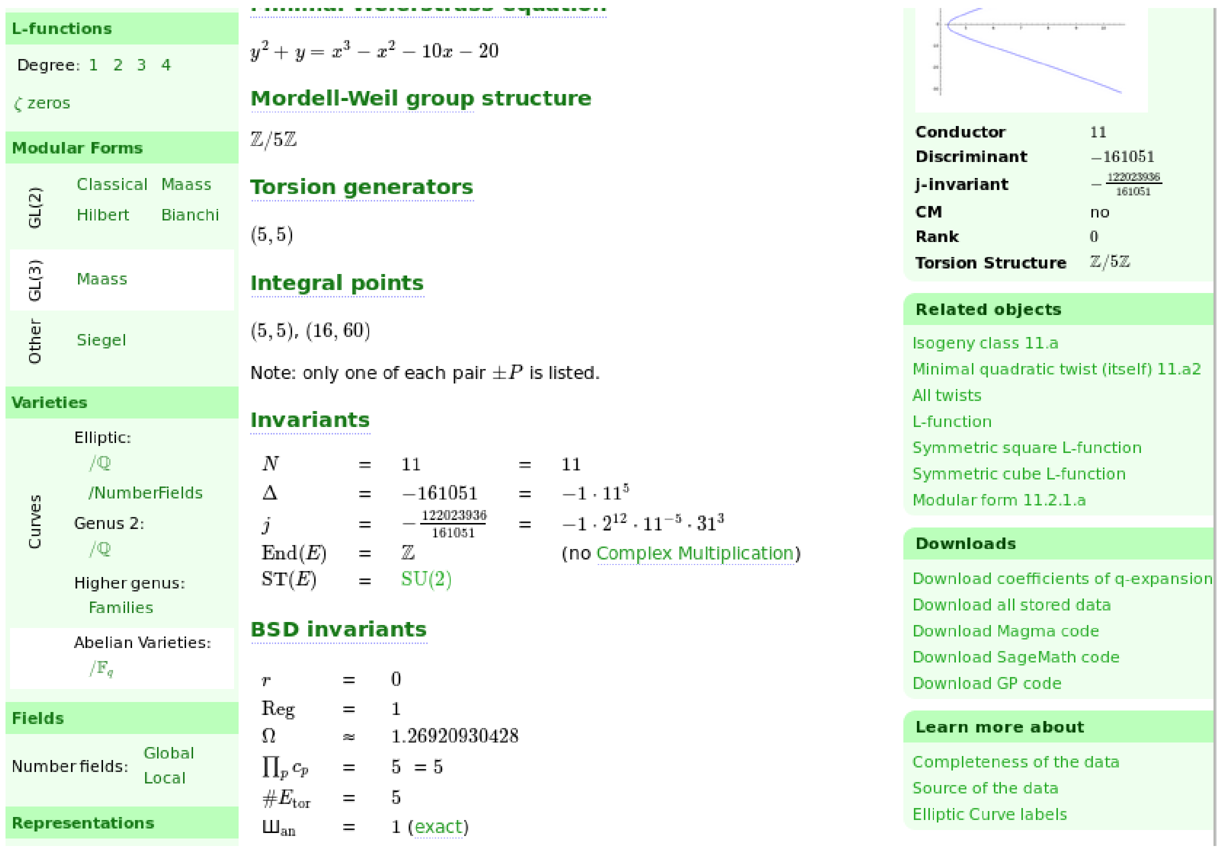
3.1. History

The idea behind Sage-Explorer originated at a workshop organized by Paul-Olivier Dehaye and the second author at ICMS in 2013. The aim of this workshop was to bring together developers of online mathematical databases like the LMFDB and computational mathematical software like SAGE.

LMFDB is an online database in number theory for L-functions and Modular Forms. One key asset, beyond advanced searches, is the ability to display objects together with a lot of contextual information, such as relevant invariants, related objects, and associated mathematical knowledge; see Figure 10. For each type of mathematical object the content of the page is handpicked by experts, which ensures a highly relevant selection. However this process does not scale well when the variety of objects increases. Also, the page can only show precomputed data.

Computational systems like Sage on the other hand let's one compute with a wide variety of objects; semantic information embedded into the system makes this scalable by enabling generic code. However, in the usual Read-Eval-Print loop of a computational system, objects are displayed without contextual information. Most of the information is readily available, typically through introspection; however that introspection needs to be carried out manually which hampers fast pace interaction and discoverability.

One of many projects at that workshop was to explore whether one could bridge some of the gap between the two worlds and get the best of both. Namely, given an object in a computational system, exploiting the semantic information embedded into the system to generate automatically a page that displays it together with a relevant selection of contextual information. Of course,



L-functions

Degree: 1 2 3 4

ζ zeros

Modular Forms

GL(2): Classical Maass Hilbert Bianchi

GL(3): Maass

Other: Siegel

Varieties

Curves: Elliptic: /$\mathbb{Q}$ /NumberFields Genus 2: /$\mathbb{Q}$ Higher genus: Families

Abelian Varieties: /$\mathbb{F}_q$

Fields

Number fields: Global Local

Representations

$y^2+y=x^3-x^2-10x-20$

Mordell-Weil group structure

$\mathbb{Z}/5\mathbb{Z}$

Torsion generators

$(5,5)$

Integral points

$(5,5)$, $(16,60)$

Note: only one of each pair $\pm P$ is listed.

Invariants

N	=	11	=	11
Δ	=	-161051	=	$-1\cdot 11^5$
j	=	$-\frac{122023936}{161051}$	=	$-1\cdot 2^{12}\cdot 11^{-5}\cdot 31^3$
$\mathrm{End}(E)$	=	$\mathbb{Z}$		(no Complex Multiplication)
$\mathrm{ST}(E)$	=	SU(2)		

BSD invariants

r	=	0
Reg	=	1
Ω	≈	1.26920930428
$\prod_p c_p$	=	5 = 5
$\#E_{\mathrm{tor}}$	=	5
Ш$_{\mathrm{an}}$	=	1 (exact)

Conductor	11
Discriminant	-161051
j-invariant	$-\frac{122023936}{161051}$
CM	no
Rank	0
Torsion Structure	$\mathbb{Z}/5\mathbb{Z}$

Related objects

Isogeny class 11.a
Minimal quadratic twist (itself) 11.a2
All twists
L-function
Symmetric square L-function
Symmetric cube L-function
Modular form 11.2.1.a

Downloads

Download coefficients of q-expansion
Download all stored data
Download Magma code
Download SageMath code
Download GP code

Learn more about

Completeness of the data
Source of the data
Elliptic Curve labels

FIGURE 10. The LMFDB page for the elliptic curve "11a2".

one can't hope to beat with a generic solution a thoughtful selection made by experts of the field; nevertheless this may be compensated by the large coverage of types of objects.

A proof of concept was implemented by Jason Bandlow and the second author in a matter of days. The implementation was very crude and not sustainable due to the use of web technologies that did not meld well with SAGE. Nevertheless, it showed that the concept was meaningful; more importantly, it led to the discovery that this approach invited users to go on a journey, exploring SAGE by jumping from objects to related objects, hence the name, SAGE-Explorer.

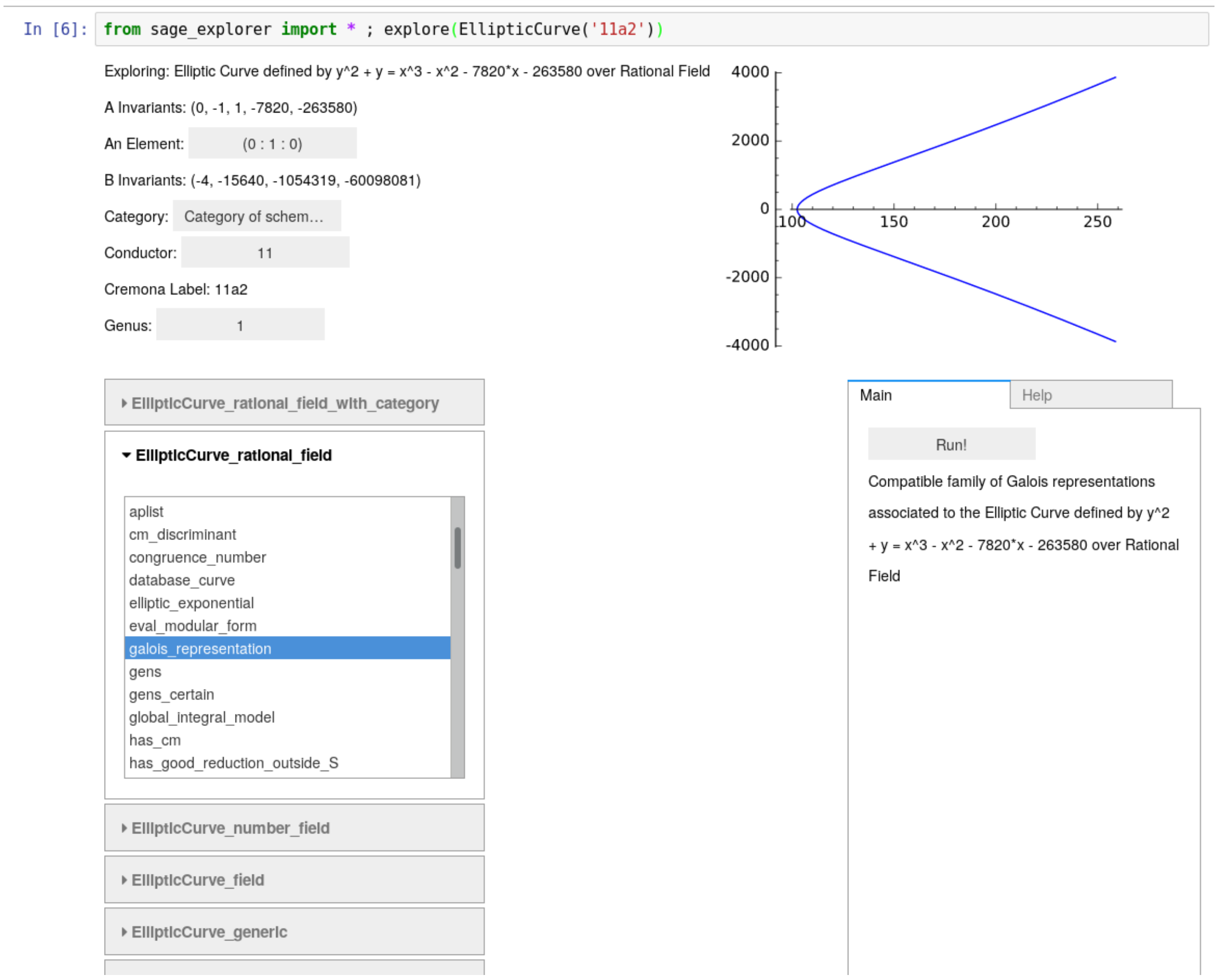
3.2. A JUPYTER-based SAGE-Explorer

The emergence of the JUPYTER and JUPYTER widget technology at the time of writing OpenDreamKit's proposal opened an opportunity for reimplementing SAGE-Explorer on top of a sustainable foundation. The current implementation started in June 2018 and took a few weeks. For any SAGE object o in a Jupyter session, running `explore(o)` opens a page with the following contextual information:

- a representation of the object, as text, latex formulae, graphics, or interactive widget depending on availability;
- a list of operations (methods) available for the object, sorted by the class providing that operation, with the ability to swiftly lookup the documentation or run the operation, possibly providing additional arguments;
- a selection of *properties*, that is relevant invariants or related objects.

See Figure 11 for the page produced by Sage-Explorer on the same elliptic curve as above.

Most objects appearing on the page are clickable to start exploring them, with history management. The displayed properties are selected according to the semantic attached to the object. For



```
In [6]: from sage_explorer import * ; explore(EllipticCurve('11a2'))
```

Exploring: Elliptic Curve defined by y^2 + y = x^3 - x^2 - 7820*x - 263580 over Rational Field

A Invariants: (0, -1, 1, -7820, -263580)

An Element: (0 : 1 : 0)

B Invariants: (-4, -15640, -1054319, -60098081)

Category: Category of schem…

Conductor: 11

Cremona Label: 11a2

Genus: 1

Compatible family of Galois representations associated to the Elliptic Curve defined by y^2 + y = x^3 - x^2 - 7820*x - 263580 over Rational Field

FIGURE 11. Sage-Explorer on the elliptic curve “11a2”

example, if the object is a finite set, its cardinality will be displayed. This is currently configured by a text file in *YAML* syntax a short extract of which follows.

```
cardinality:
  in: EnumeratedSets.Finite

characteristic:
  in: Fields

genus:
  isinstance: sage.schemes.elliptic_curves.ell_generic.EllipticCurve_ge

multiplication_table:
  in: Semigroups.Finite
  when: cardinality < 21
```

Finally, calling `explore()` without arguments brings up an index page from which the user can explore Sage’s catalogs of graphs, groups, algebras, crystals, etc.

3.3. **Dissemination and future plans**

The code is distributed as a Sage package sage-explorer, endowed with a Binder-based online demo.

At this stage, Sage-Explorer is still a prototype. It needs to be battle field tested for stability and usability. We will seek for feedback from users, in particular for styling, for growing the properties configuration beyond the current minimal seed, and for exploring how to best empower them to customize themselves this configuration. We will also improve the existing features, enabling for example searches and categorizing of operations, and exploring how to categorize properties as is done on LMFDB pages.

Last but not least, we will investigate the interplay of Sage-Explorer with WP6's Math-in-the-Middle approach. Recall that this approach is is to exploit, combine, and enrich the semantic information contained in computational systems and databases, for interoperability purposes and beyond. One of the ongoing plans in this work package is to embed additional semantic information – like alignments with the central Math-in-the-Middle ontology – in SAGE and other computational systems, for example through code annotations. Albeit informal, whether a given property is "interesting" for a given type of objects is semantic information; as such it could be included, either in the central ontology or in the systems.

We will explore whether the Math-in-the-Middle interoperability layer can be used to:

- share the configuration of interesting properties?
- explore other other computational systems from Sage?
- derive native explorers for other systems with minimal custom code for each system?

Disclaimer: this report, together with its annexes and the reports for the earlier deliverables, is self contained for auditing and reviewing purposes. Hyperlinks to external resources are meant as a convenience for casual readers wishing to follow our progress; such links have been checked for correctness at the time of submission of the deliverable, but there is no guarantee implied that they will remain valid.