Online databases: from L-functions to combinatorics

organized by
Paul-Olivier Dehaye and Nicolas M. Thiery

Workshop Summary

1 Introduction

This workshop brought together two mathematical communities. The first is the LMFDB collaboration, where LMFDB stands for *L-functions and modular forms database*. The second is the sage-combinat community, which is concerned with combinatorics. While these two collabortations study very different mathematical objects, to some extent they both rely on Sage, an open-source computer algebra system. They also represent two of the strongest areas of expertise of Sage.

In furthering its research goals, each community had previously developed its own set of software tools. For instance, the *category* framework is used by sage-combinat for structuring code, while the LMFDB relies on a custom interactive, web-based presentation of precomputed number theoretic objects and their relations.

The purpose of the workshop was to evaluate the suitability of these software tools in the other community and generally raise awareness of existing work that could be reused or developed jointly.

2 Lectures

The morning lectures at the workshop served as introductions to existing projects.

findstat.org, lmfdb.org: both mathematical database projects in the Sage ecosystem, in combinatorics and number theory respectively. These were presented by Christian Stump and David Farmer. Findstat only represented a small part of the effort tied to sage-combinat.

category framework: an implementation in Sage of basic categorical information, intended to help structure code (presentation by Nicolas Thiéry).

Astronomical Data Center (Strasbourg): introduction by Francoise Genova to data centers in astronomy and their development.

cubicweb: a highly-customizable software tool designed to organize and manage databases, according to principles of the semantic web. This was developed as open-source software in python by the French company Logilab. The presentation was done jointly by Nicolas Chauvat and Florent Cayré.

These lectures lead to several different projects, taking place the rest of the time.

3 Projects

3.1 New page in the LMFDB on crystals

Anne Schilling designed a webpage to present computations on crystals. These computations are done using algorithms she had previously implemented in Sage (with the help of others). The work done by Schilling, with the help of Harald Schilly, consisted of calling these Sage functionalities from the framework currently used by the LMFDB. This is now accessible at

http://testing.modform.org/Crystals/

3.2 New page in the LMFDB on permutations

Similarly, Sébastien Labbé exploited the LMFDB templates to create new pages on permutations, this time at http://www.lmfdb.org/Permutations/ .

Individual permutations have human-readable URLs, such as http://www.lmfdb.org/Permutations/s

In addition, his work was done in coordination with Schilling's: they extracted the common base code for the page of a generic object on the LMFDB, which will help refactoring the LMFDB code efficiently.

3.3 Refactorization of code related to modular forms in the LMFDB

Stephan Ehlen and Fredrik Stromberg worked together on refactoring LMFDB code related to modular forms, building an abstraction layer above the various databases already present in the LMFDB. This will help integrate more data about modular forms in the future.

Together with Hertmut Monien, they improved the LMFDB homepages of congruence subgroups, which now show generators of the group and a "nicer" picture of the fundamental domain.

They also reported a Sage bug to Sébastien Labbé (trac #13998), who fixed it during the week. This bug affected thousands of pickled objects in the LMFDB database.

3.4 Further development of knowls

Harald Schilly has previously implemented the idea of knowls for the LMFDB project. These allow for easy inclusion of explanations to the various objects that are presented, and help avoid the "dry" aspect of many mathematical databases. On the other hand, they are not intrusive so they do not interfere with an expert user. While their look and feel might seem familiar, some features make them very distinct. For instance, they are meant to convey essential bits of information, which can then be combined together (by nesting). This builds up documentation in a very modular fashion, fitting very well with constructions usually seen in mathematics.

The consensus in the LMFDB collaboration is that this tool would be very helpful for other websites related to mathematics.

During the workshop, Harald Schilly improved the implementation of knowls, separating it from the rest of the code of the LMFDB. This facilitates further reuse of this idea.

3.5 Sage Explorer prototype

Sage Explorer is a tool for exploring Sage objects and connections between them. It displays a Sage object, some relevant information about this object, and links to related objects

(those that can be obtained using a method of the object). One central feature of the tool is to make it easy to configure which piece of information is relevant, typically depending on the semantic of the object.

https://github.com/jbandlow/sage-explorer
https://explore.sagemath.org (some day)

More information is available on the README and TODO there.

The current prototype was implemented by Jason Bandlow and Nicolas M. Thiéry. The purpose was to evaluate whether such a tool could be written as a thin view layer above Sage, and how much the semantic information available in Sage was useful and sufficient for that purpose. One of the specific experiment was to essentially reproduce the web page of an elliptic curve over the rationals (with the help of David Farmer). On the other hand, the prototype did not attempt to tackle the following aspects:

- Accessing / searching / selecting in the database / creation of forms
- Beautifying the site with CSS style configuration
- Appropriate queries to retrieve Sage objects (at this point, they are encoded as Sage expressions in the url).

3.5.1 Conclusions

At this point, the semantic information already available in Sage (essentially in the hierarchy of classes and categories) did the job. Further potentially useful semantic information includes an appropriate hierarchy of categories for elliptic curves, systematic implementation of the construction method that describes how an object can be reconstructed, and (rich) signatures for methods/morphisms. However, for the later, one would need to experiment with strong use cases in order to specify precisely the needs and choose an appropriate design (syntax for the annotations, annotations only on abstract methods or also on all their implementations, ...).

The prototype is indeed a thin layer: 300 lines of infrastructure code, 150 lines of html templates, 60 lines of config. As desired, the infrastructure is completely generic and allows for exploring any Sage object. The object-dependent configuration (what are the important properties that should be displayed) is quite concise, and does not depend on the rendering detail (no html). It actually could be used as is to build an heavy-weight application instead of a web interface. The only prerequisite for a contributer to expand it is to know the mathematics of the object to be displayed and how to call the appropriate Sage commands to compute its properties.

Altogether the participants seem to have appreciated the demonstration as convincing.

Sage Explorer has been released publicly on github and advertised on the Sage developers mailing list for someone to pick it up. The authors will maintain it and foster the transition, but won't take the lead of its long term further development. After some polishing, and with an appropriate infrastructure to run it securely, Sage Explorer should be run on a public server (explore.sagemath.org?), and advertised to a wider audience to draw attention and man power.

The two remaining questions are:

- Is the Sage Explorer a tool worth developing further?

- Should the LMFDB infrastructure be refactored, either using Sage Explorer or a some similar approach?

To answer these questions, one needs to assess how the prototype would scale, and whether the community has the required will and manpower.

For the first point, a good 2-3 day project for a small team would be to take a section of the LMFDB and try to reproduce it within Sage Explorer. The infrastructure part will need to be extended to handle searches and queries, and to beautify the result. The metric of success will be whether it remains completely generic, and whether the config file remains concise and easy to write and maintain. One challenge is to design layout-control idioms that remain simple and rendering independent, while keeping a beautiful on-screen rendering. The other challenge is the handling of the interaction with databases.

3.6 Database for invariant rings of permutations groups

Nicolas Borie had a prototype of database containing information about invariant rings of permutations groups he calculated systematically during his PhD thesis on computational invariant theory. He rewrote that prototype using the cubicweb framework with the help of Nicolas Chauvat from Logilab.

The result is convincing on the principle: with very little information (a few dozen lines) given in a semantic schema, the site could offer at once an interface that covered all features of the previous database and quite more. However the learning curve is still steep without the direct help of one of the developers. Hence, for a small project like this one, the result is to be put into balance with the efforts.

3.7 Cubicweb based findstat.org

Similarly, Christian Stump used the cubicweb framework to work on a prototype second version of his website findstat.org.

3.8 Database of parents and beyond

A recurrent need for Sage user is to find all the implementations in Sage of objects having a given structure. For example, one would want to know all the (type of) groups that are implemented in Sage.

Maintaining a list of implementations by hand would not be practical, given the number of implementations and of types of structures.

Florent Hivert implemented a prototype for this feature using the following fact: the Sage coding conventions state that at least one instance of each class should be tested using the generic test suite (see TestSuite). Therefore, by instrumenting TestSuite and running all the Sage tests, one can extract a pickle of at least one instance of each class, and index those instances according to their category. One can also extract the construction used for this instance.

Of course, the quality of the result depends directly on the systematic application of the coding conventions. However, all in all, the result is certainly more robust than any hand-generated index.

This feature would typically be used by the Sage Explorer to display all the known implementations of a category.

3.9 Other projects

- Parallel compilation of the Sage documentation (Hivert, Hansen)
- Finalization of the Sage interface to the Coxeter3 library (Schilling, Hansen)
- Improvements to the pages related to elliptic curves and number fields on lmfdb.org (for instance, factorization of Dedekind zeta functions into Artin L-functions: Cremona, Jones)
- Systematic computation of symmetric squares of elliptic curve L-functions (Bober)
- (Partial) review of trac tickets (Thiéry, Hivert, Chapoton, Hansen, Labbé, Cohen and many others)

4 Suggestions for future projects

4.1 Progressively migrate the model from LMFDB into Sage

4.1.1 Code

Ideally, all the code handling mathematical calculations should be implemented in Sage, and just called by LMFDB. Here are some examples of features that could be moved from LMFDB to Sage:

- Hierarchy of classes for L-functions, and methods from the associated objects (elliptic curves, number fields, Dirichlet characters,...) A good part of the design of this hierarchy should be fairly straightforward thanks to the discussions between the experts.
- Conversion code between different labelling schemes: for elliptic curves (partly integrated in Sage already), for Dirichlet characters (patch waiting), for number fields,...
- Code to compute the Γ-factors of symmetric squares of L-functions, factorizations of Dedekind zeta functions, ...

A good way to locate what needs to be migrated into Sage would be to go through the pages of the LMFDB, and try to reproduce them in the Sage Explorer and see what is missing.

An issue might be the overhead induced by the Sage workflow. This could be minimized by writing the code as a library outside of the Sage sources, using MonkeyPatch'ing whenever adding methods to existing classes. Then, the code could be merged into Sage by chunks while it stabilizes.

4.1.2 Database

Similarly, all database queries should ideally be implemented in Sage. The benefits would be:

- A clear separation of concerns between the view and model.
- All features directly available to a light weight web interface such as the Sage Explorer.
- The possibility to have hybrid methods that transparently attack the database or (re)compute data, using (shared) persistent caching. A typical candidate would be the modular_degree method for elliptic curves.

- The ability to run such queries and searches pragmatically from Sage, using either a local database or the LMFDB database.
- A good step toward using the LMFDB as a web service.

4.2 Persistent and shared caching

Use case: Macdonald and k-Schur functions and in general symmetric functions are graded Hopf algebras. When doing research on those, a key tool is to look, for example, at the transition matrices between two basis at a given degree. As the degree increases, those matrices get expensive to compute. It's therefore desirable to store them in a database between Sage sessions, or even better to share them between several users/computers. Both for user's convenience and for usage as a building block for higher level methods, it's very desirable that this process be seamless: namely that one always calls the same method which either fetches the data, if already available, or triggers a calculation (with addition to the database).

Idiom: Sage has a very simple to use idiom for adding (non persistent) caching properties to a method of function cached_method, which has proven very practical to this respect.

Goal: implement a decorator, similar to cached_method, that turns a usual method into a method with a persistent cache, typically in the form of a key-value database on the disc (mongodb?). Then, later calls to this method with the same arguments, even in a separate Sage session would fetch the data from the database.

There already exists a decorator for this in Sage (sage.misc.func_persist.func_persist). It needs to be put to heavy use and further developed. Some potential features include:

- Easy user configuration (using monkey patching of some sort) of which methods/functions should be persistently cached, without changing the Sage sources (this feature would also be desirable, though less critical, for cached methods).
- Easy transfer / synchronization of the database between machines for a given user (note: plain file databases have the nice feature that one can trivially sync them with rsync, but that might not be compelling reason).
- Sharing persistent caches between users, typically using a shared read-write server. Whenever a new value is computed by some user, this new value makes its way, directly or in batches, into the server. External computation bots may also progressively fill in the database.

  This involves important security issues (the data in a cache could be a pickle; reconstructing a malicious pickle may trigger the execution of arbitrary code on the users' machine). Therefore, unless restricted to small group of users which trust each other, there needs to be a well defined security policy. This should typically include some reviewing process. One could also envision adding an entry to the database if at least, say, three independent users calculated the same value.

  As was clear from Florence Genova's presentation, the traceability of data is central here, both as a mean to discourage the introduction of malicious code, and to be able to withdraw malicious, corrupted, incorrect, or outdated data from the database in the event that a source is deemed untrusted (malicious user, incorrect computations, incompatible versions of Sage, ...).

4.3 Interactive graphical widgets for combinatorial objects

An important need for combinatorics is to visualize graphically combinatorial objects and interact with them. The current infrastructure has proven to be either inadequate or too complicated. Latex output is beautiful but static and fragile; Sage's interact objects are intrinsically too slow in their current implementation, and limited to the notebook. Deploying new interactive components in the notebook requires to master too many techniques (Sage+Javascript+AJAX+...) for the average Sage contributor (see the graph_editor). Anne Schilling's experiment with the Littelmann path crystal viewer shows that the LMFDB framework makes it easier but still involves too much training and overhead to make it likely for many Sage contributors to jump in and implement interactive graphical widgets for their pet objects.

A very interesting project is Larch Env (https://sites.google.com/site/larchenv/). It's an IPython notebook, designed from the ground up to allow for both textual and graphical interaction, and for the easy implementation of new object-specific interactive widgets which can be combined together as a lego.

A key feature would be to implement backend-independent widgets that would work both in heavy weight applications like the iPython or Larch Env notebooks or in web applications (using AJAX).

5 Conclusion

Some of the projects have shown that there was an overlap between sage-combinat and LMFDB interests. For instance, the work of sections and reused the LMFDB framework to present combinatorial objects (knowls, homepage per object, browsing and searching features,...). Conversely, Sage Explorer used some of the ideas of the category framework to emulate the presentation of the LMFDB for number-theoretic objects in a more automatic way. Another set of project explored the use of cubicweb for similar purposes.

Each project offered a different solution to the problem of interacting with the result of mathematical computations. These computations could be performed live or stored in a database, while the interaction could be searching or clicking via a web browser, or queries performed from more specialized clients (for instance Sage itself). The solutions that were presented are not mutually exclusive.

The workshop has not lead to a unique superior design: while theoretically possible, the variety of needs and desires that were expressed at the workshop would lead to something that everyone sees as too complicated. A down-to-top approach is favoured, even if it might require some unification work later.