The OpenDreamKit Project: Towards Enhanced Interoperability via a Math-in-the-Middle Approach

Michael Kohlhase[1] Nicolas M. Thiéry[2]

[1] Jacobs University, Bremen, Germany
[2] Université Paris-Sud, Paris, France

Abstract. OpenDreamKit— "Open Digital Research Environment Toolkit for the Advancement of Mathematics" — is an H2020 EU Research Infrastructure project that aims at supporting, over the period 2015–2019, the ecosystem of open-source mathematical software systems, and in particular popular tools such as LinBox, MPIR, SageMath, GAP, Pari/GP, LMFDB, Singular, MathHub, and the IPython/Jupyter interactive computing environment. From that ecosystem, OpenDreamKit will deliver a flexible toolkit enabling research groups to set up Virtual Research Environments, customised to meet the varied needs of research projects in pure mathematics and applications.
An important step in the OpenDreamKit endeavor is to foster the interoperability between a variety of systems, ranging from computer algebra systems over mathematical databases to front-ends.
In this paper, we describe the OpenDreamKit project and report on experiments and future plans with the *math-in-the-middle* approach. This information architecture consists in a central mathematical ontology that documents the domain and fixes a joint vocabulary, combined with specifications of the functionalities of the various systems. Interaction between systems can then be enriched by pivoting off this information architecture.

1 Suggested restructuring of *Integrating Mathematical Software Systems via the MITM approach* section

MMT system
GAP
- Brief introduction to the GAP type system
- Tentative approaches to exporting GAP types
- n application: consistency checker for the GAP documentation

SAGE
- Short description of Sage's category system
- Tentative approach to formalize Sage's categories in MMT
- An application: multisystem semantic handle interfaces

LMFDB
- Short description of LMFDB's goals and implementation
- Tentative approach to formalize LMFDB semantics in MMT

2 Towards a Mathematical Virtual Research Environment

From their earliest days, computers have been used in pure mathematics, either to make tables, to prove theorems (famously the four colour theorem) or, as with the astronomer's telescope, to explore new theories. Computer aided experiments, and the use of databases relying on computer calculations such as the Small Groups Library in GAP, the Modular Atlas in group and representation theory, or the LMFDB, are now part of the standard toolbox of the pure mathematician, and certain areas of mathematics completely depend on it. Computers are also increasingly used to support collaborative work and education.

The last decades witnessed the emergence of a wide ecosystem of open-source tools to support research in pure mathematics. This ranges from specialized to general purpose computational tools such as GAP, PARI/GP, LinBox, MPIR, Sage, or Singular, via online databases like the LMFDB. Not counting of course online services like the Wikipedia, arXiv, or MathOverflow. A great opportunity is the rapid emergence of key technologies, and in particular the Jupyter (previously IPython) platform for interactive and exploratory computing which targets all areas of science.

This has proven the viability and power of collaborative open-source development models, by users and for users, even for delivering general purpose systems targeting a large public (researchers, teachers, engineers, amateurs, ...). Yet some critical long term investments, in particular on the technical side, are in order to boost the productivity and lower the entry barrier:

- Streamlining access, distribution, portability on a wide range of platforms, including High Performance Computers or cloud services.
- Improving user interfaces, in particular in the promising area of collaborative workspaces as those provided by SageMathCloud.
- Lowering barriers between research communities and promote dissemination. For example make it easy for a specialist of scientific computing to use tools from pure mathematics, and reciprocally.
- Bringing together the developers communities to promote tighter collaboration and symbiosis, accelerate joint development, and share best practices.
- Outsourcing as much of the development as possible to larger communities to focus the work forces on their core specialty: the implementation of mathematical algorithms and databases.
- And last but not least: Promoting collaborations at all scales to further improve the productivity of researchers in pure mathematics and applications.

These can be subsumed by the goal of *Virtual Research Environments* (VRE), that is online services enabling groups of researchers, typically widely dispersed, to work collaboratively on a per project basis. This is exactly where the OpenDreamKit project kicks in:

2.1 OpenDreamKit

The project "Open Digital Research Environment Toolkit for the Advancement of Mathematics"' [ODKb] is a European H2020 project funded under the

EINFRA-9 call [EI] whose theme of *Virtual Research Environments* was a natural fit to seek for manpower and funding for a developer community that have been working really hard on the items above. The OpenDreamKit consortium consists of core European developers of the aforementioned systems for pure mathematics, and reaching toward the numerical community, and in particular the Jupyter community, to work together on joint needs. The project aims to help address the following aims in close collaboration with the community:

1. Further improve the productivity of researchers in pure mathematics and applications by further promoting collaborations on *Data*, *Knowledge*, and *Software*.
2. Make it easy for teams of researchers of any size to set up custom, collaborative *Virtual Research Environments* tailored to their specific needs, resources, and workflows.
3. Support the entire life-cycle of computational work in mathematical research, from *initial exploration* to *publication*, *teaching*, and *outreach*.

The acceptance of the proposal [ODKa] in May 2015 was a strong sign of recognition, at the highest level of funding agencies, of the values of open science and the strength and maturity of the ecosystem.

The OpenDreamKit projects [ODKa] will run for four years, starting from September 2015. It involves about 50 people spread over 15 sites in Europe, with a total budget of about 7.6 million euros. The largest portion of that will be devoted to employing an average of 11 researchers and developers working full time on the project. Additionally, the participants will contribute the equivalent of six other people working full time. By definition this project will be mostly funding actions in Europe; however those actions will be carried out, as usual, in close collaborations with the worldwide community (potential users of the VRE as well as developers outside the OpenDreamKit consortium).

An innovative aspect of the OpenDreamKit project is that its preparation and management happens, as much as is practical and without infringing on privacy, in the open. For example, most documents, including the proposal itself, are version controlled on public repositories and progress on tasks and deliverables is tracked using public issues (see [ODKb]). This has proven a strong feature to collaborate tightly with the community and get early feedback.

2.2 Work plan

In practice, the OpenDreamKit work plan consists in 58 concrete tasks split in seven work packages, which include:

WP3: Component Architecture work on portability – especially on the Windows platform – modularity, packaging, distribution, deployment, standardization and interoperability between components.

WP4: User Interfaces e.g. work on uniform Jupyter notebook interface for all interactive computational components, improvements toJupyter, 3D visualization, documentation tools, ...

WP5: High Performance Mathematical Computing e.g. work within and between ODK's components to improve performance, and in particular better exploit multicore / parallel architectures.

WP6: Data/Knowledge/Software-Bases e.g identification and extensions of ontologies and standards to facilitate safe and efficient storage, reuse, interoperation and sharing of rich mathematical data whilst taking into account of provenance and citability; data archiving and sharing in a semantically sound way component architecture; Integration between computational software and databases;

WP7: Social Aspects research on social aspects of collaborative software/-data/knowledge development in mathematics to inform the

OpenDreamKit will also actively engage in community building and training by organizing workshops and training materials. BOP:1

2.3 Existing Virtual Research Environments for Mathematics

Early VRE's include SageMathCloud, a collaborative online environment where students, teachers and researchers can create, customize, and share a project. This project essentially consists of a virtual machine, with a simple web-based user interface, and ready-to-use software for interactive computations (e.g. Sage) and authoring (e.g. LaTeX), with facilities for real-time communication through chat, video, and shared editing of documents, programs and worksheets. For education purposes, course material can be provided as worksheets, assignments can be distributed, collected, and returned as well.

Technically speaking, SageMathCloud is a specific open-source cloud-based Virtual Research and Teaching Environment for mathematics developed since 2013 under the lead of William Stein, with funding from the NSF, and Google's Education Grant program. [2] [3] EdN:2 EdN:3

It presently hosts over 250,000 projects and has over 12,000 weekly active users. This fast adoption by a wide variety of users demonstrates the relevance and the long-term impact this kind of collaborative environments can have. EOP:1

3 Integrating Mathematical Software Systems via the Math-in-the-Middle Approach

To achieve the goal of assembling the ecosystem of mathematical software systems in the OpenDreamKit project into a coherent mathematical VRE, we have to make the systems interoperable at a mathematical level. In particular, we have

[1] Old Part: MK: this should shortened, also we must mention Mathematica, Maple, what else? What are their common parts?

[2] EdNote: NT: Mention SageMath, Inc.?

[3] EdNote: SL: 2016: SageMath, Inc. accepted in Google startup program, see https://twitter.com/wstein389/status/708439137200836608

to establish a common meaning space that allows to share computation, visualization of the mathematical concepts, objects, and models (COMs) between the respective systems. Building on this we can build a VRE with classical IDE techniques.

Concretely, the problem is that the software systems in OpenDreamKit have their own representations of and functionalities for the COMs involved. This starts with simple naming issues (e.g. elliptic curves are represented by ec in the LMFDB, and as EllipticCurve in SageMath), persists through the underlying data structures, and becomes virulent at the level of algorithms, their parameters, and domains of applicability.

[4] EdN:4

3.1 An application: toward multi-system semantic aware handle interfaces

The handle paradigm in system interfaces The "handle" paradigm has become a classic when interfacing two computational mathematics systems. For example, most of the SAGE interfaces, including that for GAP, SINGULAR, or PARI use this paradigm to delegate calculations to those systems.

In this paradigm, when a system `A` delegates a calculation to a system `B`, the result `r` of the calculation is not converted to a native `A` object; instead `B` just returns a handle (or reference) to the object `r`. Later `A` can run further calculations with `r` by passing it as argument to `B` functions or methods. Advantages of this approach include:

- Avoiding the overhead of back and forth conversions between `A` and `B`.
- Manipulating objects of `B` from `A` even if they have no native representation in `A`.

Semantic handle interfaces Whenever `A` and `B` share some common semantic (for example the concept of group), it's desirable that handles behave as native `A` objects. For example, if a group `G` is constructed in `B`, one wants to manipulate handles to `G` or its elements as if they were native `A` groups or group elements, even if there is no corresponding native implementation for `G` in `A`. This can be achieved with the usual *adapter* design pattern. The bulk of the work is the implementation of adapter methods so that, for example, calling the method `h.cardinality()` on a SAGE handle `h` to a GAP object `G`, triggers in GAP a call to `Size(G)`.

In SAGE, this has been implemented in a couple special cases. For examples, SAGE permutation groups or matrix groups are built on top of handles to GAP objects. However, this implementation is monolithic and does not scale. For example, if `h` is a handle to a set `S`, SAGE only knows that `h.cardinality()` can be computed by `Size(S)` in GAP if `S` is a group; in fact if `h` has been constructed through the `PermutationGroup` or `MatrixGroup` constructors. Whereas we would want this method to be available as soon as `S` is a set.

[4] EDNOTE: MK: essentially the St. Andrews stuff

Generic/hierarchical semantic handle interfaces During the first joint GAP-Sage days, the last author worked on a prototype of generic semantic handle Sage-GAP interface. The idea is twofold:

1. Every Sage category (e.g. the category of sets, of groups) can provide a collection of adapter methods that are valid for every handle to a GAP object in the corresponding mathematical category. This applies as well to elements and morphisms.
2. When a handle `h` to a GAP object `S` is created, the properties of `S` (its GAP categories and properties) are explored, and the handle `h` is then put in the matching (or closest matching) Sage category.

For example, here is the adapter for the cardinality method and some context around:

```
class Sets: # Everything about sets in Sage
    class GAP: # The adapter methods relevant to Sets in the Sage-Gap interface
         class ParentMethods: # Adapter methods for sets
             def cardinality(self): # The adapter for the cardinality method
                 return self.gap().Size().sage()
         class ElementMethods: # Adapter methods for set elements
             ...
         class MorphismMethods: # Adapter methods for set morphisms
             ...
```

At the current stage of the implementation, a handle to a GAP field behaves essentially like a native Sage field. This remains valid for objects of all subcategories as well, from magmas to rings. The infrastructure is relatively lightweight, and can be extended by developers and users as the need for more adapter methods arises.

Scaling to multisystem interfaces? A second stage was initiated during the Knowledge representation in mathematical software and databases workshop organized at the University of St Andrews, St Andrews, 25th-27th January, 2016.

The approach described earlier is likely to work well for implementing an interface between two systems. However it does not scale for interfacing `n` systems, as this requires the implementation of `n(n-1)` independent adapter interfaces.

The key point here is that implementing an adapter method (or function) typically requires only some simple abstract information on the method, namely its signature and its names in both systems. In particular, the only things that changes between an `A->B` adapter method and the equivalent `C->D` adapter method are the names of the methods.

The second stage of this project is therefore to explore whether the interfaces could be automatically generated from a consistent formalizations of the systems.

EdN:5 [5]Update this paragraph w.r.t. the rest of this section

[5] EdNote: NT

Ideally, the mathematical structure and operations would be described once, e.g. in the MMT language (the blue blob in Michael's talk) and then each system would be formalized by specifying how the operations are implemented (the purple blobs). For example, one would specify in MMT that a magma is a set with a binary operation denoted by default `o`. The relevant category in Sage is `Magmas()`, and the binary operation is implemented by the method `_mul_`.

We experimented with doing this formalization using lightweight annotations in the Sage source code such as:

```
@semantic(mmt="sets")
class Sets:
    class ParentMethods:
         @semantic(mmt="o", gap="Size")
         @abstractmethod
         def cardinality(self):
             r"""
             Return the cardinality of ``self``.
             """
```

Note: the only additions to the original source code are the `@semantic` lines.

Several variants of the annotations exist to allow for adding annotations on existing categories without touching their file, and also for specifying directly the corresponding method names in other systems when this has not yet been formalized elsewhere. Similarly, one could provide directly the signature information in case that is not yet modelled in MMT.

Difficulties In Sage and GAP (and most other systems with some category mechanism) we distinguish additive magma and multiplicative magma, duplicating all the information, code, etc. In MMT however, thanks to morphisms which allow to rename operations transparently, there is no such distinction: there are just Magmas.

Hence, to actually map additive magmas in Sage to additive magmas in GAP (and map the corresponding methods), one need in the intermediate MMT step to keep an extra bit of information, namely whether the monoid is additive or multiplicative (or something else; think of the bracket operation of Lie algebras).

3.2 Exploring GAP types

Brief introduction to GAP types and categories. [6] EdN:6

[6] EdNote: MP: I am not sure what I claim below about Sage is true, it also irks me a bit that we seem to conflate the idea of a type system with the idea of organising mathematical hierarchies. Of course in GAP system this is intentional, in Sage, I don't know. In my head Sage uses whatever python uses as the type system (duck typing?) and then intro- duces a category system on top. We should agree on a level of description that fits.

While the SAGE type system is inherently object-oriented, the GAP type system puts more of an emphasis on *operations* on and between objects. Also a feature of the GAP type system is that it tries to model the way mathematicians think about their objects of study. Breuer and Linton describe the GAP type system in [**breuer-linton**], and the GAP documentation [] has an extensive technical description.

Categories in GAP are a little bit like categories in category theory: Mathematically similar objects are in the same category, so for instance there are categories `IsSemigroup`, `IsMonoid`, and `IsGroup`, and the fact that a monoid is a semigroup, and a group is a monoid is encoded by subcategories.

Representations of objects give a way to express that for example groups can be represented as permutation groups, matrix groups, or finitely presented groups, or even more fine-grained there could be permutation groups that have a more efficient representation if they act on a small numbers of points.

An *attribute* in GAP is a value attached to a GAP object, for a group this can be its size, [7]. EdN:7

A *property* is an attribute that can only be true or false.

The values of attributes and properties can also be unknown on creation, can be computed on demand, and their values can then be stored for later reuse without the neeed to be recomputed.

The final concept in GAP to be introduced here is the concept of a *family*. Families partition the set of all objects, and control how different objects can interact. For instance all permutations in GAP are in one family, and multiplication is only defined between objects in the same family. [8]. EdN:8

EdN:9 [9]

EdN:10 [10]

Tentative approaches to exporting GAP types. Encoded in the categories, representations, attributes, and properties in GAP there is a wealth of mathematical knowledge. GAP allows some introspection of this knowledge after the system is loaded.

Having a clear picture of the relations between different objects is very helpful to GAP developers, package authors, and users. One might be interested in the attributes or properties that GAP can compute for an object, or how it tries to compute them.

During the OpenDreamKit workshop in St Andrews in January 2016 we developed some tools to more conveniently access mathematical knowledge encoded in GAP's object system and export this information in several ways:

– introspection inside a running GAP session

[7] EDNOTE: todo find some attributes that groups can have
[8] EDNOTE: todo: this is not entirely true, but do we care?
[9] EDNOTE: NT: It could be informative to add an example of category and property. For example Magma and Associative. Or Set and Finite. Ideally, we would use the same example in the GAP and Sage section.
[10] EDNOTE: TODO (???) Compare and contrast it with the Sage type system

– export to a simple structured format such as JSON, for use with MMT
– export as a graph for visualisation for exploration

[11] EdN:11

We will make these prototype implementations available as part of the standard GAP distribution. We hope that they will be also enhanced by the GAP Jupyter interface developed in `https://github.com/gap-packages/jupyter-gap`.

As a side-effect of the work outlined above, we fixed a number of minor bugs in the installation of special categories in GAP.

The JSON output of the GAP object system after loading a default set of packages is currently around 11 Megabytes in size and takes many hours to import into MMT.
[12] [13] [14] [15] EdN:12 EdN:13 EdN:14 EdN:15

An application: consistency checker for the GAP documentation. One of the immediate outcomes of the development of the tools described in the previous section is the consistency checker for the GAP documentation.

GAP uses special format for its main manuals. It is called GAPDoc and is provided by the GAP package with the same name [**gapdoc**]. Besides main manuals, it is adopted by 97 out of 130 packages currently redistributed with GAP. Using GAPDoc, one builds text, PDF and HTML versions of the manual from a common source given in XML.

GAPDoc defines XML constructions to specify the type of the documented object (function, operation, attribute, property, etc.). However, due to the limitations of the semi-automated conversion of GAP manuals from the TEX-based manuals used in GAP 4.4.12 and earlier, a number of objects had their types stated incorrectly.

We developed the consistency checker for the GAP documentation, which extracts type annotations from the documented GAP objects and compares them with their actual types. It immediately reported almost 400 inconsistencies out of 3674 manual entries. In the subsequent cleanup, we by now have eliminated about 75% of them. The consistency checker will appear in the next release of GAP 4.8.3, and will be available via `make check-manuals`. It also performs other useful checks: for example, it produces a list of manual sections having no

[11] EdNote: picture based on `https://github.com/OpenDreamKit/OpenDreamKit/issues/165`?

[12] EdNote: MP: Actually, we have only exported the type information that is loaded when starting GAP with the default loaded packages, there is a lot more in the additional packages we could load

[13] EdNote: MP: What about consistency checks within teh GAP object system?

[14] EdNote: NT: Do you have anything to say about the GAP-MMT formalization? E.g. hints on potential ways this formalization may be written?

[15] EdNote: MP: I have some ideas as to how I would write an MMT formalisation, unfortunately I do not understand MMT well enough yet to know whether my ideas make any sense. I'll think about this a bit more and add my comments then

examples. Thus, the new tool helps to improve the quality of GAP documentation, and may be useful for the similar checks of those GAP packages which use GAPDoc-based manuals.

3.3 Presenting LMFDB data

Short description of LMFDB's goals and implementation The *L-functions and modular forms database* is a project involving dozens of mathematicians, who assemble computational data about L-functions, modular forms and related number theoretic objects. The main output of the project is a website, hosted at `http://www.lmfdb.org`, that presents this data in a way that could serve as a reference for research efforts and should be accessible at the graduate student level. The mathematical concepts underlying the LMFDB are extremely complex and varied, so part of the effort has been focused on how to relay mathematical definitions and their relationships to data and software. For this purpose, the LMFDB has developed so-called *knowls*, which are a technical solution to present LaTeX-encoded information interactively, heavily exploiting the concept of transclusion. The end result there is a very modular and highly interlinked set of definitions in mathematical natural language.

The LMFDB backend is written in python, with a heavy reliance on Sage and the document database system MongoDB [**lmfdb-repo**]. Again, due to the complexity of the objects considered, many idiosyncratic encodings are used for the data. This makes the whole data management lifecycle particularly tricky, and dependent on different select groups of individuals for each component.

Tentative approach to MMT semantic layers for the LMFDB Since the LMFDB spans the whole "vertical", from writing software, to producing new data, up to presenting this new knowledge, it is a perfect test case for a large scale try-out of the "knowledge in the middle": a semantic layer would be beneficial to its activities across data, knowledge and software, which it would help integrate more cohesively and systematically. Among the components of the LMFDB, elliptic curves stand out in the best shape, and a source of best practices for other areas. For this reason, the OpenDreamKit collaboration targeted that relatively small subset of the LMFDB for its prototype. We plan to extend coverage in a second phase, and expect it to be relatively easy if we manage to demonstrate the added value of a semantic layer for our prototype.

In practice, our first step was to integrate into the MMT system the mathematical knowledge that had already been made explicit in the knowls (see Fig. 1). For this, we used the sTeX interpreter, which was able to crudely type-check the definitions, avoiding circularity and ensuring some level of consistency in their scope. Knowledge translated into this format immediately became browsable through MathHub.info, a project developed in parallel to MMT to host such formalisations.

The second step consisted of translating these informal natural language definitions into progressively more exhaustive MMT formalisations of mathematical concepts (see Fig. 2). This is possibly the hardest step, as that layer will

```
\begin{mhmodnl}{elliptic-curve}{en}
  \begin{definition}[id=ec.q.def]
    An \defii{elliptic}{curve} over a \trefi[field]{field} $K$ is a smooth projective
    curve $E$ defined over $K$ of genus $1$ together with a (distinguished) $K$-rational
    point $O$ (called the \defii{base}{point}) of $E$.
  \end{definition}

  \begin{assertion}[type=remark]
    The most commonly used model for \trefis{elliptic}{curve} is a plane cubic defined by
    a \trefi[Weierstrass-model]{Weierstrass}{equation}, with the \trefii{base}{point}
    being the unique point at infinity\ednote{reference}.
  \end{assertion}

  \begin{assertion}[type=remark]
    \guse{minimal-Weierstrass-model} Note that such an equation is not unique over
    $\RationalNumbers$. $E$ has a unique reduced
    \trefiii[minimal-Weierstrass-model]{minimal}{Weierstrass}{model}.
  \end{assertion}
\end{mhmodnl}

\begin{mhmodnl}{minimal-Weierstrass-model}{en}
  \begin{definition}[id=ec.q.minimal-Weierstrass-model.def,
   title=Minimal Weierstrass model of an elliptic curve of the $\RationalNumbers$]
    Every \trefii[elliptic-curve]{elliptic}{curve} over $\RationalNumbers$ has an integral
    Weierstrass model (or equation) of the form
    \[\minwe xy{\livar{a}1}{\livar{a}2}{\livar{a}3}{\livar{a}4}{\livar{a}6}\]
    where $\livar{a}1,\livar{a}2,\livar{a}3,\livar{a}4,\livar{a}6$ are integers.  Each
    such equation has a \trefi[discriminant-weierstrass-equation]{discriminant}
    $\wediscriminantOp$.

    A \defiii{minimal}{Weierstrass}{model} is one for which
    $\absolutevalue\wediscriminantOp$ is minimal among all Weierstrass models for the same
    curve.  For elliptic curves over $\RationalNumbers$, minimal models exist, and there
    is a unique minimal model which satisfies the additional constraints
    $\minset{\livar{a}1,\livar{a}3}{\set{0,1}}$, and $\inset{\livar{a}2}{\set{-1,0,1}}$.
    This is defined as the reduced minimal Weierstrass model of the elliptic curve.
  \end{definition}
\end{mhmodnl}
```

Fig. 1. S
ample sTEX implementation of LMFDB knowl content for elliptic curves and their Weierstrass models.

have to eventually interoperate with the informal sTeX definitions, the data that has been previously computed, the software to compute more, and the software needed to view all this information. We found that the best approach was to push for ”small theories”, again emphasizing modularity.

```
namespace http://www.opendreamkit.org/elliptic_curves D
theory Weierstrass_model : odk:?Math =
  Weierstrass_model : tp D
  polynomial_equation : tm Weierstrass_model → tm polynomial D
  polynomial_equation_injectivity :
    A, B ⊢ polynomial_equation A ≐  polynomial_equation B → ⊢ A ≐  B D
  discriminant : tm Weierstrass_model → tm int D
  a_invariants : tm Weierstrass_model → tm (vector int 5) D
  a_invariants_factors_injectivity :
    A,B ⊢ a_invariants A ≐ a_invariants B →⊢ A ≐ B D
M
theory elliptic_curve : odk:?Math =
  include ?Weierstrass_model  D
  elliptic_curve : tp  D
  base_field : tm elliptic_curve → field D
  Weierstrass_model_construction : tm Weierstrass_model → tm elliptic_curve D
  minimal_Weierstrass_model : tm elliptic_curve → tm Weierstrass_model D
  construction_surjective:
    A ⊢ Weierstrass_model_construction (minimal_Weierstrass_model A) ≐ A  D
M
theory minimal_Weierstrass_model : odk:?Math =
  include ?elliptic_curve D
  minimal : tm Weierstrass_model  →  tm Weierstrass_model D
  is_minimal : tm Weierstrass_model  →  prop O = [A] (minimal A) ≐ A D
  minimality_idempotence : A ⊢ minimal (minimal A) ≐ minimal A D
  minimality_of_minimal_Weierstrass_model :
    A ⊢ is_minimal (minimal_Weierstrass_model A)  D
  injective_minimal_Weierstrass_model :
    A,B ⊢ minimal_Weierstrass_model A ≐ minimal_Weierstrass_model B  → ⊢ A ≐ B  D
M
```

Fig. 2. S
ample MMT formalisation of elliptic curves and their Weierstrass models.

The next step was in integrating computational data into MMT. We first needed to expand the capabilities of MMT with respect to literals. For this purpose, the MMT core team constructed an MMT interface with the LMFDB API, which delivered data in JSON format. This is not fully satisfactory, and a more generic solution could be found, interfacing for instance directly with the MongoDB database. Once JSON documents were obtained, we still needed to define explicit specifications for them. There are recent ongoing efforts [**lmfdb-formats**] to document in natural language the formats used by the LMFDB, and this provided an easy starting point. Actually, the scope of those specifications had to be extended, as they were not focused only on the format used to store the data fields (string *vs.* float *vs.* integer tuple, for instance), but also the types these data fields were meant to represent (rational *vs.* polynomial, for instance). It turns out that many patterns exist there: often different people will encode the same semantic type with the same data format, following the same conventions. Therefore, to maximize reusability, we again emphasized modularity and developed *MMT codecs*. Codecs integrate *some* information about semantics and some about data formats, and are composable. For instance, one could compose two MMT codecs, say *polynomial-as-reversed-list* and *rational-as-tuple-of-int*, to signify that the data $[(2,3),(0,1),(4,1)]$ is meant to represent the polynomial $4x^2+2/3$. Of course, these codecs could be further decomposed (signalling which variable name to use, for instance). The initial costs of developing these codecs are high, but the clarity gained in documentation is valuable, they are highly reusable, and they drastically expand the range of tooling that can be built around data management. Our efforts also fit neatly alongside similar efforts underway across the sciences to standardize metadata formats (for instance through the Research Data Alliance's Typing Registry Working Group[**rda-typing**]).

4 Conclusion

[16] EdN:16

Acknowledgements The authors gratefully acknowledge discussions and experimentation at the St. Andrews workshop, which clarified the ideas behind the math-in-the-middle approach; in particular John Cremona, Paul-Olivier Dehaye, Luca de Feo, Mihnea Iancu, Alexander Konovalov, Samuel Lelièvre, Steve Linton, Dennis Müller, Markus Pfeiffer, Viviane Pons, Florian Rabe, and Tom Wiesing.

We acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541) and from the EPSRC Collaborative Computational Project CoDiMa (EP/M022641/1).

References

[EI] *EINFRA-9: e-Infrastructure for Virtual Research Environment.* URL: `http://ec.europa.eu/research/participants/portal/desktop/en/opportunities/h2020/topics/2144-einfra-9-2015.html`.

[16] EdNote: MK: I am not sure we need this, but we will see

[ODKa] *Open Digital Research Environment Toolkit for the Advancement of Mathematics*. Project Proposal. URL: `https://github.com/OpenDreamKit/OpenDreamKit/raw/master/Proposal/proposal-www.pdf` (visited on 09/01/2015).

[ODKb] *OpenDreamKit Open Digital Research Environment Toolkit for the Advancement of Mathematics*. URL: `http://opendreamkit.org` (visited on 05/21/2015).