REPORT ON OpenDreamKit DELIVERABLE D6.8
Curated Math-in-the-Middle Ontology and Alignments for GAP/ SAGE/ LMFDB

JOHN CREMONA, DENNIS MÜLLER, MICHAEL KOHLHASE, MARKUS PFEIFFER, FLORIAN RABE, NICOLAS M. THIÉRY, TOM WIESING

Due on	1/09/2018
Delivered on	4/09/2018
Lead	Friedrich-Alexander Universität Erlangen/Nürnberg (FAU)
Progress on and finalization of this deliverable has been tracked publicly at: `https://github.com/OpenDreamKit/OpenDreamKit/issues/142`	

ABSTRACT. Work Package **WP6** develops a novel, foundational, knowledge-based framework for interfacing existing open source mathematical software systems and knowledge bases into a mathematical VRE, where systems can delegate functionalities among each other seamlessly without losing semantics.

The overall Math-in-the-Middle (MitM) Framework developed in **WP6** over the last three years is described in **D6.5**; this Report complements it by describing the curated contents Math-in-the-Middle (MitM) Ontology which serves as a reference and pivotal point for translations between the various input languages of mathematical software systems and knowledge bases.

In a nutshell, the MitM Ontology describes the mathematical objects, concepts, and their relations in a general, system-agnostic way in an OMDoc/MMT theory graph while the mathematical systems export API theories that describe the system interface language in terms of types, classes, constructors, and functions – again in OMDoc/MMT. These two levels of descriptions are linked by OMDoc/MMT alignments that allow the translation of expressions between systems.

Together with deliverable report **D6.5**, this report describes the implementation and initial evaluation of the MitM integration and interoperability paradigm initially envisioned in deliverables **D6.2** and **D6.3**. The MitM paradigm constitutes the core development goal of **WP6** and the curated content described in this report enables running non-trivial integration case studies. In the future we hope to further consolidate content, increase coverage of alignment, and greatly extend the reach of the integration both in terms of OpenDreamKit systems covered as well as knowledge available in the MitM Core ontology.

Contents

1. Introduction 3
2. The MitM Ontology 4
2.1. The MitM Foundation 4
2.2. MitM Core 4
2.3. The “Semantic Multilingual Glossary of Mathematics” (SMGloM) 5
3. The OpenDreamKit System APIs 6
3.1. Generated System API Theories 6
3.2. The MitM Alignments 6
4. Evaluation & Conclusion 7
References 9
Appendix A. Types & Expressions in the MitM Foundation 10
A.1. Base Types 10
A.2. Complex Types 10
A.3. Types for Mathematical Structures and their Models 10
A.4. Subtyping 11
A.5. Expressions over MitM Foundation Types 11

1. Introduction

The Math-in-the-Middle (MitM) Ontology is developed in the OpenDreamKit project as a central information resource for the Math-in-the-Middle (MitM) approach to flexible and knowledge-based integration of mathematical software systems. It serves as a reference and pivotal point for translations between the various input languages of mathematical software systems. This integration and interoperability has been described in [Deh+16; WKR17; Koh+17] and – in great detail – in [D6.518]. In a nutshell, the MitM Ontology describes the mathematical objects, concepts, and their relations in a general, system-agnostic way in an OMDoc/MMT theory graph while the mathematical systems export API theories that describe the system interface language in terms of types, classes, constructors, and functions – again in OMDoc/MMT. These two levels of descriptions are linked by OMDoc/MMT alignments [Mül+17b] that allow the translation of expressions in the interface language of system A into the MitM-induced language, and from there to the interface language of system B [Mül+17a].

The Math-in-the-Middle Ontology is hosted on the MathHub.info system [Ian+14; MH], the sources can be obtained from `http://gl.mathhub.info`. As the MathHub front-end is currently undergoing major re-write[1] we will reference the MitM ontology by its sources – the semantically enhanced user interface can be accessed via the same URLs without the `gl.` part.

The development consists of four parts:

i) The **MitM ontology** which has the formalization of mathematical background theories which expresses the knowledge in terms of mathematical concepts found in the mathematical literature, without concern – and thus abstracting from – for system requirements.

ii) The **SMGloM Glossary** which has a human-oriented – and thus informal –, but semantically structured versions of the same content, and can therefore act as a human-readable documentation.

iii) **System API theories for the OpenDreamKit Systems and mathematical data bases**, these express the system-specific instances of the mathematical concepts, constructors, procedures, and types that make up the input/output language of the respective systems.

iv) The **Alignments** between the OpenDreamKit System API theories and the MitM ontology.

We will discuss all four parts separately below, and conclude with a self assessment of the MitM at this stage.

[1]The old MathHub interface was based on Drupal, which led to major system vulnerabilities and therefore maintenance hassles, because Drupal was targeted by hackers. We are currently working on a docker-based orchestration of services with a React.JS based front-end in the general spirit of the OpenDreamKit VRE toolkit; see `https://github.com/MathHubInfo/`. The new system can be accessed at `http://new.mathhub.info`.

2. The MitM Ontology

In this section, we describe the parts of the MitM ontology, which is a formal development in the MMT system. The formalizations can be found at `https://gl.mathhub.info/MitM/`. That location also contains various other archives in the `MitM` library, which are experiments, where the MitM ontology has been picked up by other projects.

The MitM ontology consists of two parts: the **MitM Foundation** which provides the logical language in which the mathematical domains in the MitM ontology can be described, and the **MitM Core**, which consists of the knowledge about these domains represented in the MitM foundation.

2.1. The MitM Foundation

The MitM foundation provides the type system and logic for the MitM ontology, i.e., the basic representational infrastructure used in the MitM ontology. It is developed in the archive `Foundation` (4 files, 539 LoF (lines of formalization), 103 commits).

Types & Expressions. While the logic is relatively straightforward, the standardization of the type system was very difficult because mathematics requires a rich, open-ended type system, yet concrete implementations must stay as simple as possible. After surveying the OpenDreamKit systems, we developed the types described in Appendix A. These types come with a set of constructors that allow to specify expressions (members of the respective type) that represent mathematical objects systematically; see Appendix A.5 for an exemplary list of mathematical objects that can be represented as a members of these types.

In a nutshell, the MitM foundation provides base types for all arithmetical number systems and common data structures like strings/words and Boolean values. Complex types build on this include structural ones like (total and partial dependent) function, product, record, types, and mathematical constructions like sets, multisets, ists, vectors, and matrices. In particular, dependent record types allow to represent types for mathematical structures (e.g. rings, fields, polynomials, finite maps, etc.) and their models. These can also be constructed from OMDoc/MMT theories via the novel Mod operator [MRK18].

Finally, the MtiM foundation supports subtyping for the arithmetical number systems ($\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R} \subseteq \mathbb{C}$), and along the sub-model relation. This allows to express many mathematical identies very naturally.

2.2. MitM Core

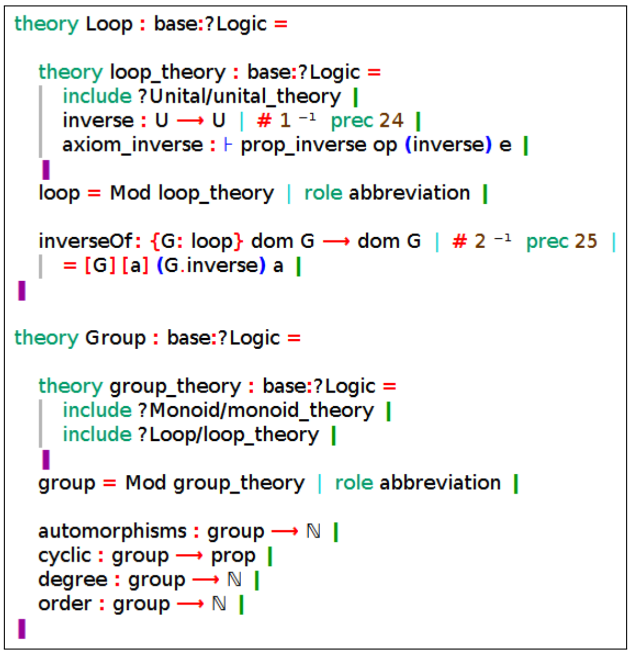
The **MitM core** in the `smglom`[2] archive (55 files, 2600 LoF, 360 commits). It carries the bulk of the knowledge representation in the MitM Ontology. The main thrust of curation has been to get the VRE use cases reported on in [D6.518], but we also have elementary formalizations of algebra, arithmetics, calculus, category theory, set collections, elliptic curves (for LMFDB), functional analysis, geometry, graph theory, measure theory, set theory, and topology.

In the sequel, we describe two exemplary parts of the MitM core ontology.

2.2.1. *Computational Group Theory.* Our formalization of CGT (part of the core ontology, found in `algebra/computational_groups`) follows the template of its implementation in GAP, and requires several levels of abstraction – currently *abstract*, *representation*, *implementation*, and *concrete*. From our experience, we expect this pattern to be applicable across computational algebra, possibly with additional levels of abstraction. The left box in Figure 3 gives an overview.

[2]The name `smglom` was initially chosen for the "Semantic Multilingual Glossary of Mathematics" (SMGloM; see 2.3 below) with which it is cross-referenced. We will rethink naming once the MitM Ontology stabilizes.

The abstract level contains the axioms and basic definitions of the theory of *Groups*: generating sets, homomorphisms, group actions, stabilisers, and orbits. The most basic part is given in Figure 1.



```
theory Loop : base:?Logic =

  theory loop_theory : base:?Logic =
    include ?Unital/unital_theory ❙
    inverse : U ⟶ U ❘ # 1 ⁻¹ prec 24 ❙
    axiom_inverse : ⊦ prop_inverse op (inverse) e ❙
  ❚
  loop = Mod loop_theory ❘ role abbreviation ❙

  inverseOf: {G: loop} dom G ⟶ dom G ❘ # 2 ⁻¹ prec 25 ❘
    = [G] [a] (G.inverse) a ❙
❚

theory Group : base:?Logic =

  theory group_theory : base:?Logic =
    include ?Monoid/monoid_theory ❙
    include ?Loop/loop_theory ❙
  ❚
  group = Mod group_theory ❘ role abbreviation ❙

  automorphisms : group ⟶ ℕ ❙
  cyclic : group ⟶ prop ❙
  degree : group ⟶ ℕ ❙
  order : group ⟶ ℕ ❙
❚
```

FIGURE 1. A Formalization of a Group

At the representation level groups are described as concrete objects suitable for computations: groups of permutations, groups of matrices, finitely presented groups, groups obtained by algebraic constructions or using polycyclic presentations.

At the implementation level, we encode implementation details: for example, permutation groups are considered as finite subgroups of the group $S_{\mathbb{N}+}$, and constructed by providing a set of generating permutations.

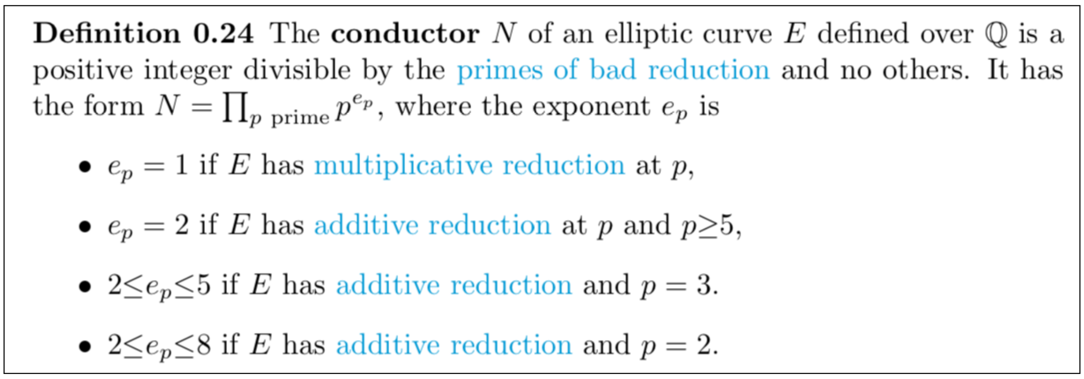
At the concrete level, the computation happens: while the higher levels are suitable for mathematical deduction and inference, this level is where OpenDreamKit systems like GAP perform their main work.

2.2.2. *Modeling and Simulation.* The `models` archive (11 files, 650 LoF, 113 commits) is an experimental extension of the MitM ontology, where we test the expressive power of MitM framework (and the OpenDreamKit technologies) by applying it to a field well outside of mathematics, namely for modeling and simulation in opto-electronics.

2.3. The "Semantic Multilingual Glossary of Mathematics" (SMGloM)

The SMGloM library is available at `https://gl.mathhub.info/smglom`. It contains ca. 815 glossary modules (OMDoc/MMT theories) with more than 1700 concepts. All represented in sTeX, a semantic variant of LaTeX developed by FAU and (earlier at) JacU. Figure 2 shows an example. The boldface words are *definienda* (i.e. concepts to be defined; here "conductor") and the blue ones are concepts already defined in other parts of the MitM Ontology. SMGloM definitions are multilingual – mostly English and German, but also some Romanian, Turkish, Arabic, and Chinese, and are cross-linked on the concept level.

The SMGloM library was started in 2013 before the OpenDreamKit project, but only enough content has been developed to fix the data model and system functionality. About half of the content of the library and all of the informal-formal alignments have been developed in response to needs by OpenDreamKit case studies.



Definition 0.24 The **conductor** N of an elliptic curve E defined over $\mathbb{Q}$ is a positive integer divisible by the primes of bad reduction and no others. It has the form $N = \prod_{p \text{ prime}} p^{e_p}$, where the exponent e_p is

- $e_p = 1$ if E has multiplicative reduction at p,
- $e_p = 2$ if E has additive reduction at p and $p \geq 5$,
- $2 \leq e_p \leq 5$ if E has additive reduction and $p = 3$.
- $2 \leq e_p \leq 8$ if E has additive reduction and $p = 2$.

FIGURE 2. An sTeX Definition of the Conductor of an Elliptic Curve

3. THE OPENDREAMKIT SYSTEM APIS

3.1. Generated System API Theories

We have generated and cross-linked system API theories for the computer algebra systems GAP, SageMath, and SINGUALAR, as well as the mathematical knowledge base LMFDB. They are generated from pre-existing sources in the systems like type information, code annotations, and systematic API documentation. OpenDreamKit deliverable report, Section 5 of **D6.5** [D6.518] has the details.

The OpenDreamKit System API Theories can be found at `https://gl.mathhub.info/ODK/{GAP,SAGE,SINGULAR,LMFDB,knowls}`. These contain:

- `GAP`: (210 Theories, 8470 Symbols[3], 64 Commits) Generated from a JSON export from the typeo system and documentation of the GAP system (after adding constructor annotations throughout the GAP source code). A theory corresponds to a source file of a GAP package, a symbol represents a GAP method or operation.
- `Sage`: (5431 Theories, 7279 Symbols, 73 Commits) Analogously generated from a JSON export. Theories correspond to Sage categories, symbols to methods.
- `Singular`: (179 Theories, 4519 Constants, 6 Commits) generate by heuristically parsing definitions, call patterns, and comment strings in the Singular C code.
- `LMFDB`: (161 Commits) Consists of schema theories for LMFDB databases (currently 5 databases covered, 182LoF) and interface theories for non-mathematical or LMFDB-specific concepts (labels, descriptors) represented therein (3 Files, 123 LoF). See Section 6 of **D6.5** [D6.518] for a discussion of schema theories and the content of this archive.
- `knowls`: (945 flexiformal Theories, ca. 900 Symbols; 10 Commits) generated from the LMFDB knowls system of interface documentations. The knowls resource is actively maintained and currently under intense review by the LMFDB community.

3.2. The MitM Alignments

The MitM alignments are available from `https://gl.mathhub.info/alignments/Public` (6 files, 1040 LoF, 240 Commits). They are represented as text files containing pairs of MMT URIs annotated with various semantic classification schemata; see [Mül+17b] for details about details on alignments and the representation format and Section 5.5 in **D6.5** [D6.518] for a discussion of the MitM-specific aspects and the provenance of the particular alignments.

As an example, Figure 3 sketches the alignments between the computational group theory ontology from above and the constructors and operations of GAP.

[3]Since the generated API theories do not come from `.mmt` source files, numbers of files and LoF are not informative metrics. Approximately, one `.mmt` file contains on average 2–3 theories, and 1–2 LoF correspond to one symbol.

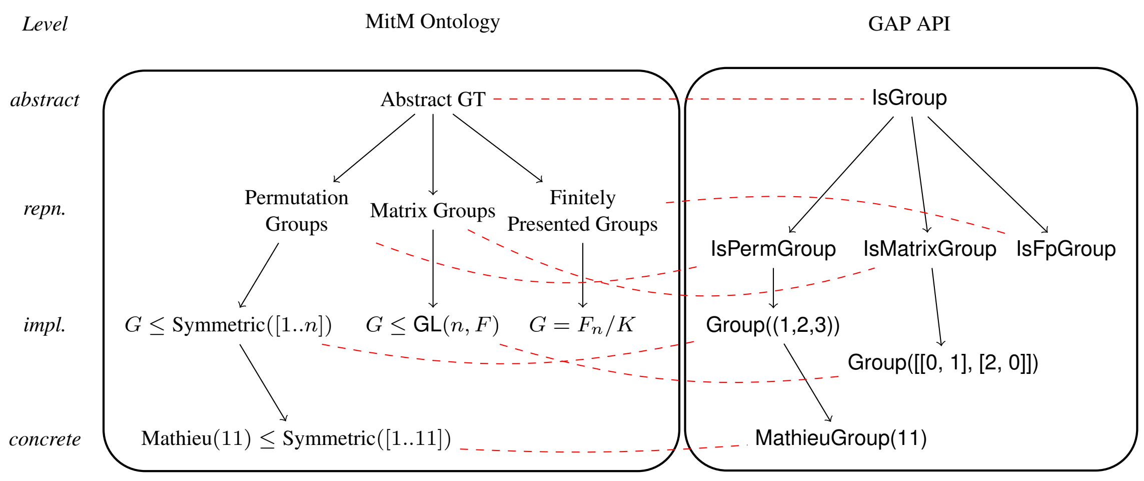


FIGURE 3. Alignments between the MitM Ontology and the GAP API

4. EVALUATION & CONCLUSION

Figure 4 shows an overview of the MitM ontology, together with the system APIs and the alignments (in red). We can directly see that the MitM ontology is dwarfed by the system API graphs. This is to be expected, since we have mainly covered the background knowledge needed for the OpenDreamKit integration use cases; see [D6.518]. This is in sync with the overall project plan, which mainly wants to demonstrate the feasibility of the MitM approach to system integration, and expects the main part of the MitM Ontology to be contributed by the mathematical community.

Most of the work in MitM Ontology has gone into provisioning a viable foundation and basic set of types, which allow the flexiformalization and specification of mathematical knowledge, objects, functions, and types in a system-independent/neutral way that closely mirrors the presentation in the mathematical literature. The central focus was in the naturalness and adequacy of the representation. The focus on a basic repertoire of types inspired by mathematical practice goes a long way into this direction.

With CGT (computational group theory) we have shown that the MitM approach is feasible, and that the investment into the extension of the MitM Ontology is commensurate with writing good (mathematical) documentation. Indeed the MitM ontology together with the system API alignments can be used for documentation purposes and complement it. Moreover, as MitM representations are shared between systems by design, all systems participating in the OpenDreamKit MitM-based integration scheme obtain this "documentation" for the cost of alignments alone.

Generally, we note that the investment necessary for adding mathematical functionality to the MitM integration decreases with the size of the existing MitM ontology, and in the limit becomes proportional to the size of the added functionality, while the amount of system functionality yielded by MitM-compatible systems increases proportionally with the MitM size. Already with four systems GAP, Sage, Singular, and LMFDB; see [D6.518] we have reached a size where investments into extending the MitM ontology are starting to pay off, and interest of the computational mathematics is generated. We expect that, once the heavy lifting of getting the framework and a seed MitM Ontology has been done, the integration framework and MitM Ontology coverage will grow organically.

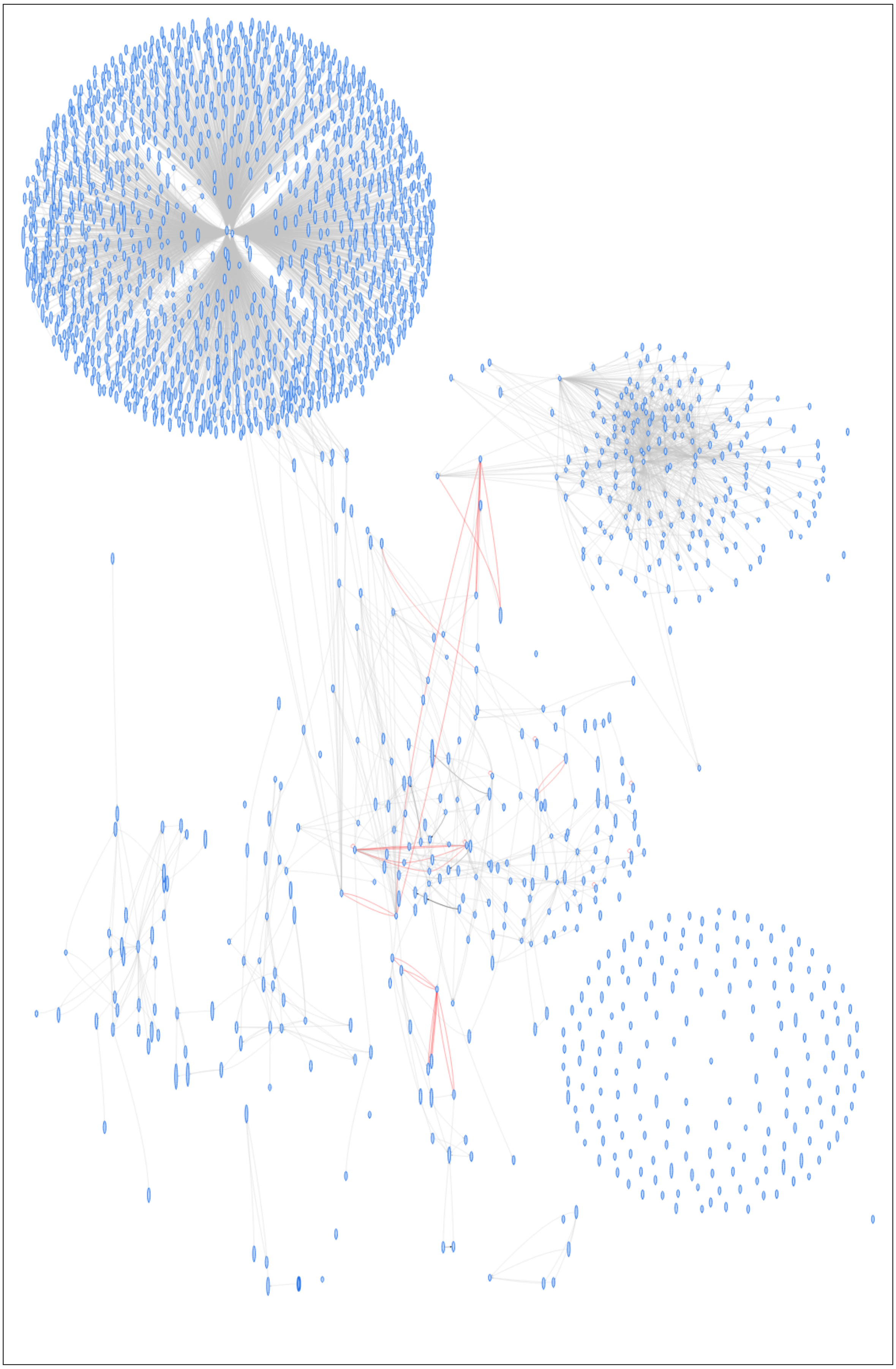


FIGURE 4. The MitM Theory Graph

References

[BKS17] Johannes Blömer, Temur Kutsia, and Dimitris Simos, eds. *MACIS 2017*. LNCS 10693. Springer Verlag, 2017.

[D6.518] John Cremona et al. *Report on OpenDreamKit deliverable D6.5: GAP/SAGE/LMFDB Interface Theories and alignment in OMDoc/MMT for System Interoperability*. Deliverable D6.5. OpenDreamKit, 2018. URL: `https://github.com/OpenDreamKit/OpenDreamKit/raw/master/WP6/D6.5/report-final.pdf`.

[Deh+16] Paul-Olivier Dehaye et al. "Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach". In: *Intelligent Computer Mathematics 2016*. Conferences on Intelligent Computer Mathematics. (Bialystok, Poland, July 25–29, 2016). Ed. by Michael Kohlhase et al. LNAI 9791. Springer, 2016. ISBN: 978-3-319-08434-3. URL: `https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/CICM2016/published.pdf`.

[Ian+14] Mihnea Iancu et al. "System Description: MathHub.info". In: *Intelligent Computer Mathematics 2014*. Conferences on Intelligent Computer Mathematics. (Coimbra, Portugal, July 7–11, 2014). Ed. by Stephan Watt et al. LNCS 8543. Springer, 2014, pp. 431–434. ISBN: 978-3-319-08433-6. URL: `http://kwarc.info/kohlhase/papers/cicm14-mathhub.pdf`.

[Koh+17] Michael Kohlhase et al. "Knowledge-Based Interoperability for Mathematical Software Systems". In: *MACIS 2017: Seventh International Conference on Mathematical Aspects of Computer and Information Sciences*. Ed. by Johannes Blömer, Temur Kutsia, and Dimitris Simos. LNCS 10693. Springer Verlag, 2017, pp. 195–210. URL: `https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/MACIS17-interop/crc.pdf`.

[MH] *MathHub.info: Active Mathematics*. URL: `http://mathhub.info` (visited on 01/28/2014).

[MRK18] Dennis Müller, Florian Rabe, and Michael Kohlhase. "Theories as Types". In: ed. by Didier Galmiche, Stephan Schulz, and Roberto Sebastiani. Springer Verlag, 2018. URL: `http://kwarc.info/kohlhase/papers/ijcar18-records.pdf`.

[Mül+17a] Dennis Müller et al. "Alignment-based Translations Across Formal Systems Using Interface Theories". In: *Fifth Workshop on Proof eXchange for Theorem Proving - PxTP 2017*. 2017. URL: `http://jazzpirate.com/Math/AlignmentTranslation.pdf`.

[Mül+17b] Dennis Müller et al. "Classification of Alignments between Concepts of Formal Mathematical Systems". In: *Intelligent Computer Mathematics (CICM) 2017*. Conferences on Intelligent Computer Mathematics. Ed. by Herman Geuvers et al. LNAI 10383. Springer, 2017. ISBN: 978-3-319-62074-9. DOI: `10.1007/978-3-319-62075-6`. URL: `http://kwarc.info/kohlhase/papers/cicm17-alignments.pdf`.

[WKR17] Tom Wiesing, Michael Kohlhase, and Florian Rabe. "Virtual Theories – A Uniform Interface to Mathematical Knowledge Bases". In: *MACIS 2017: Seventh International Conference on Mathematical Aspects of Computer and Information Sciences*. Ed. by Johannes Blömer, Temur Kutsia, and Dimitris Simos. LNCS 10693. Springer Verlag, 2017, pp. 243–257. URL: `https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/MACIS17-vt/crc.pdf`.

Appendix A. Types & Expressions in the MitM Foundation

Based on a survey of the OpenDreamKit systems, we have identified the following types for the MitM Foundation (see Section 2.1); see Appendix A.5 for an exemplary list of mathematical objects that can be represented as a members of these types.

A.1. Base Types

- number types (all unbounded in size):
  - natural numbers: N (including 0), Pos (positive, excluding 0), $Prime$ (prime numbers, not excluding 0 and 1)
  - integer numbers: Z
  - integer numbers modulo m: $Z(m)$ (for $m \in Pos$)
  - rational numbers: Q
  - real numbers: R
  - complex numbers: C
  - p-adic numbers: $Qp(p)$ (for prime p)
- strings: $String$
- booleans: $Boolean$

A.2. Complex Types

- Function types
  - the type $T_1 \to T_2$ of total functions from T_1 to T_2,
  - shallow polymorphic types $\{a_1, \dots, a_n\}T(a_1, \dots, a_n)$ for a type T with n free type variables, with the restriction that polymorphic types may not occur as subexpressions of any of the other type constructors,
- Aggregating type constructors for types $T_1, \dots, T_n$
  - product types: $T_1 * \cdots * T_n$
  - record types: $\{mpop = tfortI := a_1 : T_1, \dots, a_n : T_n mpoptImpop =, \}$ for identifiers a_i
  - disjoint union types: $T_1 + \ldots + T_n$
  - labeled disjoint union types: $[mpop = tfortI := a_1 : T_1, \dots, a_n : T_n mpoptImpop = ,]$ for identifiers a_i

  In record and labeled disjoint union types, the order of fields is not relevant.
- Collecting type constructors for any type T
  - option types (list of length up to 1): $Opt(T)$
  - vectors (fixed-length lists): $Vec(T, n)$ for $n \in N$
  - lists (arbitrary finite length): $List(T)$
  - sets (finite subsets): $FiniteSet(T)$
  - multisets (finite multiset subsets): $FiniteMultiset(T)$
  - finite hybrid sets: $FiniteHybridset(T)$ are like multisets but also allow negative multiplicities.
  - matrices: $Mat(T, m, n)$ for $m, n \in N$
- Erasure of dependent parameters: for a type constructor $T(k)$ (not necessarily unary) that takes a parameter k of number type, we write $T(\_)$ for the type obtained by taking the unions over all possible parameters. In particular, we have
  - $Mat(T, m, \_)$, $Mat(T, \_, n)$ and $Mat(T, \_, \_)$ for matrices with arbitrary dimensions
  - $List(T) = Vec(T, \_)$

A.3. Types for Mathematical Structures and their Models

- finite maps (partial functions with finite support): $FiniteMap(T_1, T_2)$ for types T_i
- polynomials: $Polynomial(r, [x_1, \dots x_n])$ for ring $r \in Ring$ and identifiers x_i
- rings: $Ring$ (multiplication is a commutative monoid)

- fields: Field
- elements of structures: every structure S (e.g., $S \in \mathit{Ring}$ or $S \in \mathit{Field}$) is a type itself; the elements of S are represented as elements of the underlying type of S, which must be defined separately for every structure S.

More generally than fixing Ring and Field, we actually allow $\mathit{Mod}(T)$ for any theory T. But the types listed above are sufficient for our purposes here.

A.4. Subtyping

The **subtyping** relation $S \subseteq T$ is the order generated by

- $\mathit{Prime} \subseteq \mathit{Pos} \subseteq N \subseteq Z \subseteq Q \subseteq R \subseteq C$
- $T(k) \subseteq T(\_)$ for any type constructor T (not necessarily unary) that allows a wildcard parameter
- All collecting and aggregating type constructors are covariant in all type arguments. For example, if $S \subseteq T$, then $\mathit{List}(S) \subseteq \mathit{List}(T)$.
- Horizontal subtyping: Record types become smaller, labeled disjoint union types bigger when adding fields.
- $\mathit{FiniteMap}$ is covariant in both type arguments.[4]
- $\mathit{Ring} \subseteq \mathit{Field}$
- For structures s, S of type T, we have $s \subseteq S$ iff $s \leq_T S$.

Here we have used the submodel order $s \leq_T S$ which holds if s is submodel of S with respect to theory T. T can be the type of models of any theory, but for our purposes, it is sufficient to consider only the case $T \in \{\mathit{Ring}, \mathit{Field}\}$. We use only the following sub-models

- $Z \leq_{\mathit{Ring}} Q \leq_{\mathit{Field}} R \subseteq\leq_{\mathit{Field}} C$
- $\mathit{Polynomial}(r, x) \leq_{\mathit{Ring}} \mathit{Polynomial}(s, y)$ if $r \leq_{\mathit{Ring}} s$ and every variable in x also occurs in y.
- If $T \subseteq T'$, then $s \leq_T S$ implies $s \leq T'S$.

A.5. Expressions over MitM Foundation Types

The MitM foundation allows the following **formulas**, i.e., expression of type $\mathit{Boolean}$:

- the propositional connectives: conjunction, disjunction, implication, negation, and equivalence of formulas as well as truth and falsity,
- typed equality: $t \doteq_T t'$ for any two terms t and t' of type T,
- typed quantifiers: $\forall x : T.F(x)$ and $\exists x : T.F(x)$ for a type T and a formula F,

Additionally, we allow shallow quantification over types: $\{a\}F(a)$ for a formula F and a type variable a is a formula but it may not — to avoid inconsistency issues — occur as a sub-formula of any of the above formula constructors.

We do not give the remaining straightforward **term** constructors in detail and only remark on a few important aspects. Most critically, we allow types to have multiple representations that are semantically equivalent but practically different in meaningful ways (e.g., because converting between representations is expensive or imprecise). Moreover, some types have multiple constructors similar to an inductive type. In the sequel, we describe which representations are supported in those cases.

Integers Modulo. The elements of $Z(m)$ are represented by $0, \ldots, m-1$.

Real Numbers. A real number can be one of the following:

- a rational number
- an IEEE double precision float
- a root $\sqrt[n]{x}$ for $n \in N$ and $x \in Z$

[4]Note that it is normal for *partial* functions to be covariant in the domain.

- the strings ”pi” and ”e”

Complex Numbers. A complex number can be one of the following:
- Cartesian form $x + yi$
- polar form $re^{i\phi}$
- root of unity ζ_n

p-Adic Numbers. A p-adic number x consists of unit $u \in N$ (u, p co-prime), valuation $v \in Z$, and precision $r \in N$ (for $u < p^r$).

Polynomials. For $r \in Ring$ and distinct strings x_i, we consider polynomials

$$p \in Polynomial(r, [x_1, \dots, x_n])$$

to be of the form $p = \Sigma_{\vec{i} \in N^n} a_{\vec{i}} \vec{x}^{\vec{i}}$ where and $(x_1, \dots, x^n)^{(i_1, \dots, i_n)}$ abbreviates $x_1^{i_1} \cdot \dots \cdot x_n^{i_n}$.

Rings. A ring can be one of the following:
- a field
- $Polynomial(r, [x_1, \dots, x_n])$ for $r \in Ring$

Fields. A field can be one of the following:
- base fields Q, R, and C
- finite fields $Z(p)$ for $p \in Prime$ (same type as integers modulo p)
- polynomial field extensions $FieldExtension(F, p, a)$ of $F \in Field$ for a polynomial $p \in Polynomial(F, [x])$ (for any variable name x)
- named fields identified by a string

We define some abbreviations for common fields:
- $Q(p, a) = FieldExtension(Q, p, a)$
- $Qsqrt(n, a) = Q(x^2 - n, a)$
- $Qzeta(n, a) = Q(y_n, a)$ where y_n is the n-th cyclotomic polynomial

We do not define $GF(q)$ for $q = p^n$ as an abbreviation for $FieldExtension(Z(p), g)$ for some irreducible polynomial $g \in Polynomial(Z(p))$ of degree n because there is no way to choose g canonically and it is necessary to know g to represent the elements of $GF(q)$.

Structure Elements. Every structure has an underlying type, which is used to represent its elements.

The underlying types of fields are defined as follows: For Q, R, C, and $Z(p)$, the underlying type is the field itself. The underlying type of $FieldExtension(F, p, a)$ is $Polynomial(F, [a])$ ($a = x$ is allowed).